SEVENTH FRAMEWORK PROGRAMME
THEME
FET proactive 1: Concurrent Tera-Device Computing (ICT-2009.8.1)

PROJECT NUMBER: 249013

Exploiting dataflow parallelism in Teradevice Computing

D7.3 – Power and Thermal Modeling and Fault-injection support

Due date of deliverable: 31st December 2011
Actual Submission: 31st December 2011

Start date of the project: January 1st, 2010 Duration: 48 months

Lead contractor for the deliverable: UNISI

Revision: See file name in document footer.

Project co-founded by the European Commission within the SEVENTH FRAMEWORK PROGRAMME (2007-2013)	
Dissemination Level: PU	
PU	Public
PP	Restricted to other programs participant (including the Commission Services)
RE	Restricted to a group specified by the consortium (including the Commission Services)
CO	Confidential, only for members of the consortium (including the Commission Services)

Change Control

Version#	**Author**	**Organization**	**Change History**
0.1	Antonio Portero	UNISI	Initial template
0.2	Alberto Scionti	UNISI	Revision
0.6	Roberto Giorgi	UNISI	Major rewriting of Section 3

Release Approval

Name	**Role**	**Date**
Antonio Portero	Originator	23.11.2011
Roberto Giorgi	State of the Art	14.12.2011
Roberto Giorgi	WP Leader	15.12.2011
Roberto Giorgi	Project Coordinator for formal deliverable	30.12.2011

TABLE OF CONTENTS

LIST OF FIGURES

Roberto Giorgi, Antonio Portero, Alberto Scionti
University of Siena

Theo Ungerer, Arne Garbade, Sebastian Weis
Universitaet Augsburg

Glossary

Auxiliary Core	A core typically used to help the computation (any other core than service cores) also referred as "TERAFLUX core"
BSD	BroadSword Document – In this context, a file that contains the SimNow machine description for a given Virtual Machine
CLUSTER	group of cores (synonymous of NODE)
COTSon	Software framework provided under the MIT license by HP-Labs
DDM	Data-Driven Multithreading
DF-Thread	A TERAFLUX Data-Flow Thread
DF-Frame	the Frame memory associated to a Data-Flow thread
DVFS	Dynamic Voltage and Frequency Scaling
DTA	Decoupled Threaded Architecture
DTS	Distributed Thread Scheduler
Emulator	Tool capable of reproducing the Functional Behavior; synonymous in this context of Instruction Set Simulator (ISS)
D-FDU	Distributed Fault Detection Unit
L-Thread	Legacy Thread: a thread consisting of legacy code
L-FDU	Local Fault Detection Unit
L-TSU	Local Thread Scheduling Unit
MMS	Memory Model Support
NoC	Network on Chip
Non-DF-Thread	An L-Thread or S-Thread
NODE	Group of cores (synonymous of CLUSTER)
OWM	Owner Writeable Memory
OS	Operating System
Per-Node-Manager	A hardware unit including the DTS and the FDU
PK	Pico Kernel
Sharable-Memory	Memory that respects the FM,OWM,TM semantics of the TERAFLUX Memory Model
S-Thread	System Thread: a thread dealing with OS services or I/O
StarSs	A programming model introduced by Barcelona Supercomputing Center
Service Core	A core typically used for running the OS, or services, or dedicated I/O or legacy code
Simulator	Emulator that includes timing information; synonymous in this context of "Timing Simulator"
TAAL	TERAFLUX Architecture Abstraction Layer
TBM	TERAFLUX Baseline Machine
TLPS	Thread-Level-Parallelism Support
TLS	Thread Local Storage
TM	Transactional Memory
TMS	Transactional Memory Support
Virtualizer	Synonymous of "Emulator"
VCPU	Virtual CPU or Virtual Core

Executive Summary

This document provides a report on the current status of the simulation platform in relation with: power, temperature and fault injection. As of our deliverable and task planning this is NOT a report of the several other activities that we carried out in this Workpackage (these are reported more synthetically in the Periodic Progress Report for Year-2). **The evidence of most of those activities is publicly available through the realized software, in particular on the public website http://cotson.sourceforge.net (branches).**

This document also serves, as of our initial intention, to document to the partners and the rest of the world:

- **the availability of some simulation features in the TERAFLUX – COTSon based simulator, to support the evaluation of Future Teradevice systems consisting of many cores (i.e., 1000+ cores), with respect to not only Performance, but also Temperature, Power, and Faults within a single toolchain that provides full-system simulation, at a reasonable simulation speed.**

As of our knowledge this is still above the State-of-the-Art when considering all the capabilities that are provided in the COTSon HP-Labs based tools and the TERAFLUX extensions.

In particular, we present the developments in Year-2 to monitor power consumption, temperature and the fault injection support model. As overall project goal, is our aim to exploit dataflow principles to reach power efficiency, reliability, efficient parallel programmability, scalability and data bandwidth. TERAFLUX project proposes the exploitation of dataflow both at the task level and inside the threads, in order to: offload and manage accelerated codes, localize the computations, manage the fault information with appropriate protocols, easily migrate code to the available/working components and **respect the power/performance/temperature/reliability envelope** for efficiently handling the parallelism and having an easy and powerful execution model, to produce a more predictable behavior. One key component to achieve that goal is the Distributed Thread Scheduler (or DTS) presented in D6.2, that operates at low-level: here we recall this, as it plays an essential role for the unified management of temperature, power, resiliency in respect to faults, and – of course – performance. The modeling in the simulator of DTS features for managing power, temperature, fault information is also detailed in this document.

IMPORTANT NOTE: This document was not strictly necessary, as the outcome that we wish to deliver was mainly SOFTWARE (this deliverable is in fact marked as OTHER not as REPORT in our Annex-1 "EC-Approved"), but we think it's anyway an useful document for the progress of the project and therefore we submit it also to point out how to use the actual software, which is the real deliverable.

1 Introduction

Our idea for taking under control the load balancing, the consumed power, the reached temperature and the resiliency to faults is to rely on a distributed resource manager, namely the Distributed Thread Scheduler (or DTS), which is available as an additional COTSon component since this Year-2 of the project on the public sourceforce.net website. The first instances are provided by HP-Labs.

An important decision, in the overall picture, regards the granularity of the above parameters that we want to control through the DTS. While many "local techniques" have been proposed we believe that future many-cores (or teradevices) will have to manage the complexity of the design by controlling such parameters at the level of the core and for 1,000 or 10,000 cores simultaneously:

"*The core is the new transistor*"

as pointed out by many. Therefore, a critical component of the new chips will be a powerful "resource manager" that can flexibly and rapidly manage the dynamic situation: our TERAFLUX architectural template embeds this important conclusion (see D6.2) and in the WP7 we are now developing the first simulative prototypes of this concept.

In the following, we explain how the work on power, temperature, faults in the WP7 is related with other WPs and previous deliverable. In the next Sections, we present a short summary of the ongoing work on power, temperature, faults in the common simulation framework.

This document also serves as part of the knowledge transfer task T7.1 and provides some "what-if" cases (requested by the Year-1 review).

1.1 *Relation to other deliverables*

Furthermore, WP7 serves to integrate the research contribution from All Partners into a single simulation platform (while keeping into account of course, any feasibility limitation). Therefore we refer to activities carried out in all other work packages. In particular:

- Work Package 2 (WP2): some applications like Graph-500 and Linpack have been tested at the 1000-core scale and we are working to make the platform encompassing also, e.g**., power, temperature** estimations. Support from BSC has been important to setup the experiments.
- Workpackage 3 (WP3): in particular support for programming models based on Transactional Memory has been proposed in the simulator by partner UNIMAN and HP. However, the integration of that work with **power and temperature** information has still to be considered.
- Workpackage 4 (WP4): preliminary back-ends form GCC have been tested and served to setup initial code testing for automatic generation of code that respects dataflow execution principle so that it will be possible the re-execute threads in presence **of faults**.
- Work Package 5 (WP5): shows model experiment techniques to detect and recover from unreliability of the system. More precisely in D5.1- Section 3 is provided the general specification of the fault detection Unit (FDU), Section 4 presents the FDU interface specification and section 5 presents the core-internal fault detection mechanism. In D.5.2, there is a section which explains a refined version of **fault detection** and recovery architecture; it reports the way a group of cores are clustered together to form a node and the

nodes are connected with a 2-D mesh based network-on-chip (NoC). There is also an explanation in inter-cluster fault detection mechanism, grouping strategies and device controller.

- Work Package 6 (WP6): provides an execution model, an architecture that uses off-the-shelf components augmented with some core extensions like the T*-ISA extension and the DTS, for reaching performance while **respecting power/thermal constraints**. In the deliverable D6.1, Sections 3 and 4, the TERAFLUX basic execution model and architecture are proposed. **The newer DTS extensions that also support information about temperature, faultiness level and power consumption.** In the deliverable D6.2, Section 2.2.1, the DTS (Distributed Thread Scheduling) is introduced: this hardware module ensures that each core remains below a predefined temperature threshold.
- Work Package 7 (WP7): D7.1 and D7.2 presented a very extensive overview of the features and the work carried out during the first year in order to enable all partners to simulate a 1000-core platform.

1.2 Year-2 activity referred by this deliverable

This deliverable reports on the research carried out in the context of Task 7.3 (m6 - 40).

An important requirement on the simulation platform for the TERAFLUX project is to provide an interface for the TERAFLUX partners to be able to support their research activities in the context of energy reduction and reliability. This involves supporting a mechanism to inject and trace faults as well as connecting the simulation platform to existing power-modeling tools. In a similar way in how we planned Task 7.2, Task 7.3 includes different stages:

- *A definition phase, where all partners together will contribute to define the requirements and specifications for the power modeling, fault injection and fault tracking interfaces. This involves defining the granularity of the power-related events required by WP3 and WP6 and - if needed - the power-impacting actuations (e.g., processor P-States, idle and sleep modes, etc.) that the research activities require. The fault-tolerant work package (WP5) will classify the types of faults we want to model, how to detect them and how to support the fault detection and recovery at the simulation level.*
- *A simulator-core implementation phase, where UNISI with guidance from HP will implement the necessary changes in COTSon to support the defined enhancements and expose them in a new version of the COTSon SDK.*
- *An application-specific implementation phase, where the interested partners will implement (test and validate) their specific fault and power models on the new SDK version and document the work to make it available to the rest of the TERAFLUX consortium.*

Hence, we believe, all goals of WP7 for the second year were achieved.

1.3 Summary of previous work (from D.7.2, D.7.1)

Most notably after the first year:

- ALL the **partners were able to use COTSon** (our common simulation framework) through a set of shared benchmarks (Project Milestone M7.1) and can commit in a single repository.

- COTSon has been **released as open-source** simulator, thus providing the reciprocal benefit for the TERAFLUX project and the international research community since the first month of the project.

- We are able to boot a simulated system including Applications, OS (full-system simulation) for the TERAFLUX Baseline Machine (TBM) as designated in D7.1 consisting of more than 1000 cores. This put us in the world-wide frontline for experimenting with simulated system of such size and it demonstrates a **continued relevance** of our objectives.

- We are able to apply modifications (e.g., we can modify the architecture of such 1000-core system). In general, we can now **integrate** the proposed research from the partners in a single platform, thus overcoming the limitation of pursuing separate research goals and rather achieving a common goal of improving such a large scale system as a whole.

- We aim at an even increased **impact** of our project using as vehicle the availability of the TERAFLUX system through the improved COTSon simulation framework. As of our knowledge, no other simulator provides similar benefits.

- We took some **extra effort** (exceeding what planned) to provide a more convincing platform for modeling a Future TERAFLUX Single-Chip/Package through the introduction of a model of an off-the-shelf network-on-chip (NoC), as we believe will benefit the whole project and the success of the simulation framework.

Regarding the interface to the FDU, in D7.1 (Section 10.3) and D7.2 (Section 3.6) we started to describe how the rest of the system reacts to the reported faults. In particular, the TSU-DTA and more recently (see D6.2) the DTS is interfaced with the Fault Detection Unit (FDU), which is in charge of reporting about the "health" of each HW component within its group of managed cores, so that the local management as well as the global resource management will be improved. Here we further detail the implementation of this interface in the COTSon simulator (in progress).

Power modeling was initially exposed by HP partner in the deliverable D7.2, Section 3.2.2. In order to show how to collect and manipulate simulation events in such a way that they can be fed to other analysis tools, such as power consumption estimation tool, HP developed a mechanism that connects the results of a COTSon simulation to the McPAT [Li09] power and area estimation tool. Here we report some practical example on how to practically use the tool in the context of TERAFLUX.

In D7.2, Section 3.4.1 is presented an instance of the model of a NoC, based on the Noxim simulator and integrated in the COTSon (see branches on SourceForge website), and it provides information about data latencies and throughput. Fault detection (partner UAU) further builds on the availability of a NoC model in COTSon.

2 Power Modeling (UNISI, HP)

2.1 Implementation of Power model in the Simulator

In order to show how to collect and manipulate simulation events in such a way that they can be fed to other analysis tools (e.g., a power estimation tool), HP developed a mechanism that connects the results of a COTSon simulation to the McPAT [Li09]. McPAT (Multi-core Power, Area, and Timing) is an integrated power, area, and timing modeling framework for CPU architectures. It simultaneously models power, area, and timing of a given CPU architecture and it supports comprehensive early stage design space exploration for processor configurations ranging from 90nm to 22nm and beyond. McPAT includes models for the components of a complete chip multiprocessor, including in-order and out-of-order processor cores, networks-on-chip, shared caches, and integrated memory controllers. It models timing, area, and dynamic, short-circuit, and leakage power for each of the device types forecast in the International Technology Roadmap for Semiconductors (ITRS) [ITRS-2009] including bulk CMOS, SOI, and double-gate transistors.

Isci et al. [Isci06] propose to use a global power management layer, which acts on a per-core basis and in coordination with higher-level scheduling and load-balancing policies set in the system software. This system is indicated to follow more properly the fast phase changes of a multi-programmed workload based on several SPEC2000 benchmarks. They consider a POWER4 simulated CMP with up to 8 cores. The DTS of TERAFLUX acts as a similar resource management and by interacting hierarchically and following the OS policies as well, but its operation is integrated also with power, temperature monitoring and detected faults.

Bergamaschi et al. [Bergamaschi08] investigate power management both at core and chip level and propose non-linear optimization algorithms acting through DVFS to control power. They use MaxBIPS algorithm (proposed by Isci et al. [Isci06]) as a reference point for their analysis. Their framework permits the evaluation of performance and power of a system encompassing cores, caches, buses, memory controller and several type of interconnection paradigms. Similarly, COTSon+McPAT allows us to evaluate both performance and power for a modular system composed of similar type of componenents. Currently, COTSon and McPAT are available as open source.

2.1.1 Definition of the granularity

We define the granularity of the minimum subsystem under study, which in our case is equal to a single core. Therefore, the set of parameters (temperature, power consumption and faultiness level) will be always considered at the core level. As explained in D6.2, these parameters are constantly considered by the Distributed Scheduler (DTS). At a smaller granularity, other techniques can of course be applied, e.g., in the functional units, cache memories and all other units that are part of a core.

2.1.2 Interface to add Power models

The COTSon simulation platform can be easily interfaced with tools for the estimation of the power consumption and the thermal behavior of a given architecture. COTSon is now including an interface to the McPAT tool.

The architecture, whose power has to be estimated, is described in McPAT through an XML input file. This input file contains both the detailed description of the simulated architecture and the list of relevant events used by the tool to generate area, power and timing estimates.

On the other hand, the COTSon is able to export simulation results (i.e., events such as instruction counts, memory accesses, etc.) in a SQLite database. In order to interface each of the two tools, a conversion tool (*cotson2mcpat*) has been implemented by partner HPLabs. The *cotson2mcpat* is a conversion tool that queries the COTSon simulation result database and generates the McPAT XML input file. In particular, the *cotson2mcpat* tool traverses the simulation database using the SQL API and queries about the configuration parameters, the simulation sampling scheme, and the individual events.

Running the McPAT tool *periodically* results in a detailed description of the power estimation for each functional unit that is described in the XML file, along with the area and timing estimations. The overall processing flow is described by Figure 1.

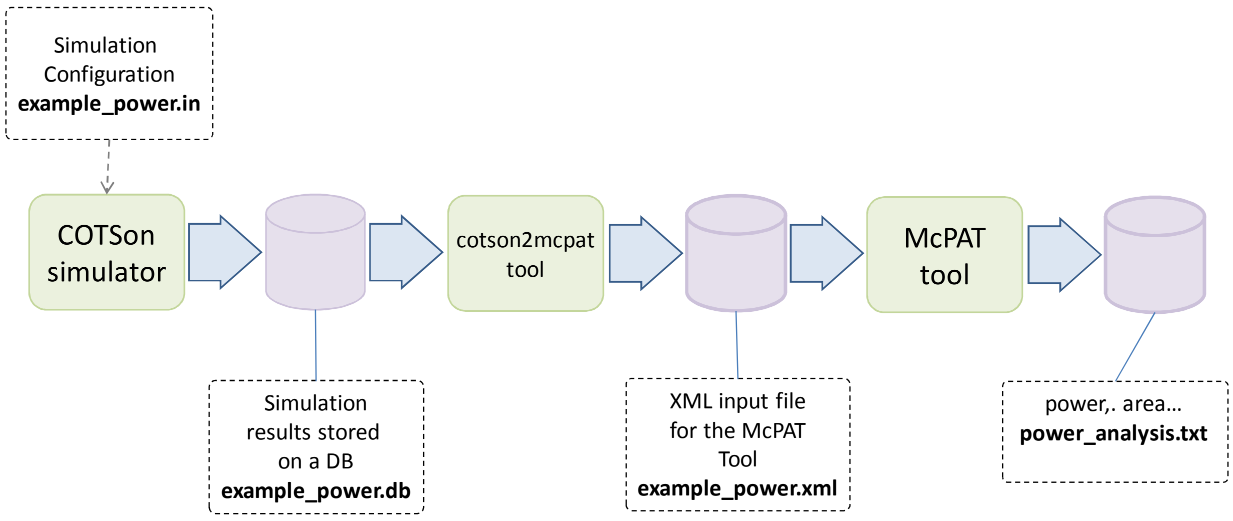

Figure 1 – Extracting power data in COTSon.

It is worth observing that the McPAT tool is currently designed to operate as *post-mortem*, but it would also be possible to invoke it during simulation (or repeatedly for individual simulation interval) to support generating power profiles over time.

Future work will consider the integration of the McPAT to dump its statistics in the database at every "heartbeat" (see below) of the COTSon simulator.

2.2 Simple Use Case 1 for power estimation

As a use case, we detail here the steps that are required to generate the power consumption estimation for a simulation of a simple architecture.

2.2.1 The simulation description

First, we start considering the input file for the COTSon simulator (in the following example: **`example_power.in`**). The detailed example and full file is reported in the Appendix 1 - Section Cotson Input File. In such file, we have to modify the following two lines:

```
[…]
use_bsd('1p-reset.bsd')
use_hdd('karmic64.img')
[…]
```

in order to correctly reflect the BSD (‘**`use_bsd`**’: the BSD it’s a “snapshot” of the simulated node at a given time, including the complete list of architectural component in a node) and the virtual disk of the virtual machine (‘**`use_hdd`**’) that we want to use for that node.

We then set the COTSon simulation parameters in a COTSon simulator (COTSon accepts as input a simple scripting language called ‘lua’). The following lines in the input file allow the simulator to store all the relevant events in a SQLite database.

In particular, the *heartbeat* parameter allows us to set the structure used to store the simulation events. In this example,

- the parameter *type* is used to set a Sqlite database (*type=”sqlite”*),
- the filename is specified by DBFILE (*dbfile=DBFILE*),
- the database structure is organized per group of experiments, thus *experiment_id* and *experiment_description* respectively set the identifier of the current experiment (i.e., group of records in the database which store the simulation events) and the associated description string.

Other general parameters are:

- The *fastforward* option, which allows us to skip the specified amount of time (expressed in nanoseconds), thus the simulation events for this period of time are not stored allowing a faster simulation (i.e. 100M ns).
- The *time_feedback* sends a time feedback to the functional simulator (e.g., AMD SimNow), so it can more accurately follow the actual simulation.
- the *max_nanos* set the maximum simulation time (nanoseconds) for the experiment.

The *sampler* parameter sets the relevant information for driving the simulation event sampling process. As the previous parameters, sampler supports several options.

- The *type* option specifies the sampling method that will be used during simulation. Setting it to *dynamic*, allows us to dynamically switching among emulation (i.e., pure functional simulation), warming phase and simulation of the target system.
- For all these phases, we can specify the amount of nanosecond (nanos) that will be used during simulation (e.g., *functional* = 500k allows us to emulate the execution of 500 microseconds). However, the max_nanos parameter specifies the maximum allowed period for the simulation. Thus both the max_nanos and the sampling intervals represent two alternative stop conditions for the simulation process.

- The *variable* option, within the sampler parameter, contains the list of simulation events that will be stored in the output log file or in the output database.
- The maxfunctional option allows us to specify the maximum time interval for the functional simulation. In particular, if the functional simulation exceeds the maximum specified value, the re-sampling procedure (i.e., the portion of the simulation used to extract the metric values) is enabled.
- Similarly, the sensitivity option is used to enable the resampling procedure whenever the normalized differential value of a monitored metric (i.e., the differential value of the metric is normalized and the difference from the average value of the metric is compared with the sensitivity) exceed the sensitivity value.

```
[…]
heartbeat = { type="sqlite", dbfile=DBFILE, experiment_id=1,
   experiment_description="test run"},
fastforward="100M",
time_feedback = true,
max_nanos="200M",
sampler=
{ type="dynamic", functional="500k", warming="100k",
  simulation="100k",
  maxfunctional=1000, sensitivity="500",
  variable={"cpu.*.other_exceptions", "cpu.*.iocount"}
},
[…]
```

2.2.2 The result database of the COTSon simulation

The simulation is typically started by invoking the **`<cotson_root>/bin/cotson`** command:

```
cotson DBFILE=\"`pwd`/example_power.db\" example_power.in
```

Where **`example_power.db`** is the name of the file which will contain the database with the COTSon simulation results, and **`example_power.in`** is the name of the input lua file (see Appendix 1 - Section Cotson Input File) for the COTSon simulator.

The Sqlite database allows us to store simulation events in a more structured format than compared to the simple log file. The database consists of 5 tables. Data resulting from the simulations are organized into experiments; hence:

- The *experiments* table stores the unique experiment identifier along with a description string.
- For each experiment, a set of rows is used to register the machine configuration (*parameters* table). Each row is composed of four fields:
    - (i) the experiment identifier,
    - (ii) the machine identifier,
    - (iii) the name of the parameter used to describe the machine configuration,
    - (iv) the value associated to the configuration parameter.
- The simulation events are stored in the database every time a heartbeat is generated (i.e., generally whenever a certain amount of time is expired, the heartbeat is generated). The *heartbeats* table registers a timestamp of each received heartbeat.

- The list of all the available target system metrics is contained in the last table, called *metric_names*.
- Finally, the *metrics* table is used by the COTSon simulator to register the simulation events effectively used among the available ones.

Whenever the heartbeat is generated, a set of rows is used for storing selected information (i.e., target system metrics) about the simulated target system (see the *variable* option of the *sampler* parameter in the COTSon input file).

2.2.3 The conversion tool cotson2mcpat

At this point it is necessary to run the conversion tool **`cotson2mcpat`**. We call this tool with the following syntax:

```
cotson2mcpat file:example_power.db 1 example_power.xml -v
```

- The input file is indicated by **`file:example_power.db`**
- The output of this step is the file **`example_power.xml`**
- Since the COTSon simulator platform organizes the database of results as a set of experiments, with all the important events recorded for each experiment, we indicated the identifier of our experiment as "**`1`**". The option '**-v**' is used to generate more information during the XML file generation process (see Appendix 1 – Section XML input file). The option '**-r**' can eventually be used to directly run the McPAT tool after the XML generation process finished.

2.2.4 The McPAT tool

McPAT is a tool designed for estimating the power consumption, the area and the timing of a microprocessor [Li09]. In order to correctly estimate these three metrics, the tool processes a XML input file containing both the description of the functional units composing the target microprocessor, and the "events" produced by an external simulator. The events concern the output metrics that the external simulator can provide during the simulation, such as instruction counting, cache memories accesses, number of simulated cycles, etc. The events and the description of the functional units are organized on a per-functional-unit basis.

To run the McPAT tool, we simply launch this command:

```
mcpat -infile example_power.xml -print_level 5 -opt_for_clk 0 >power_analysis.txt
```

- The input file is indicated with '**`-infile example_power.xml`**' (which is the output from the previous tool);
- The output of the simulation is sent to standard output (in this example, it is redirected to the file **`power_analysis.txt`**) reported in the Appendix 1 – Section McPAT simulation results.
- The option '**`-print_level 5`**' enables the McPAT tool to generate the highest level of detail in the output
- The option '**`-opt_for_clk 0`**' tunes the tool in order to consider only the ED^2P product.

Substantially, after this step we obtain a value (1 point) for the power. If we want to plot a graph of the power with time, we need to repeat this step for the next group of instructions. As mentioned, we aim to automatize this step in the future.

Finally, we expect this tool to be the foundation for the future TERAFLUX power and energy evaluations. Code and examples are available at [COTSon11].

2.2.5 From simulation to power: sample output for the L2 cache

The data from the result database after the previous simulation is converted to the following format. For example, in the following frame we highlight the data related to the L2 cache (the complete output is reported for reference in Appendix-1).

```
<component id="system.L20" name="L20">
      <param name="L2_config" value="524288,16,4,1,1,20,16,0"/>
      <param name="buffer_sizes" value="16, 16, 16, 16"/>
      <param name="clockrate" value="3200"/>
      <param name="ports" value="1,1,1"/>
      <param name="device_type" value="0"/>
      <stat name="read_accesses" value="1874"/>
      <stat name="read_misses" value="1874"/>
      <stat name="write_accesses" value="1175560"/>
      <stat name="write_misses" value="0"/>
      <stat name="conflicts" value="0"/>
</component>
```

- The *param* nodes in the above XML code portion represent the L2 cache description;
- The *stat* nodes contain the L2 cache statistics after the COTSon simulation.

The McPAT analyzes all the events associated to all the functional units and estimates the power consumption of the target system. In particular, the result of this process is an estimation of the power consumption for each functional unit in terms of

- *dynamic peak* power. This measure reports the peak power consumption during the charging and discharging of the capacitive load during normal switching activity of the functional unit.
- *gate leakage* and *sub-threshold leakage* power. This represents the power consumption due to the leakage currents in the devices forming the functional unit.
- *runtime dynamic* power. This measures the average power consumption during the charging and discharging of the capacitive load during normal switching activity of the functional unit.

The output of the previous example, in particular for the L2 cache is the following:

```
Total L2s:
  Device Type= ITRS high performance device type
    Area = 0.937911 mm^2
    Peak Dynamic = 0.165416 W
    Subthreshold Leakage = 0.217494 W
    Gate Leakage = 0.000684481 W
    Runtime Dynamic = 0.000249549 W
```

3 Thermal modeling/management (UNISI)

We decided to organize the work related to thermal modeling/management in steps. Initially, we asked to all the partners their interest in modeling the temperature inside the core: the outcome is that the TERAFLUX Consortium is more interested in the mechanisms to *manage* the temperature information ("respecting the power/performance/temperature/reliability envelope" from the abstract of our proposal – Annex-1), therefore we focused on those aspects connected to enabling a thread scheduling on the available cores based also on the additional parameter "temperature".

An important motivation for considering temperature is also that reliability and sub-threshold leakage are exponentially related to temperature.

As for the "power" parameter this will be another parameter to be considered at the scheduling time in the Distributed Thread Scheduler (DTS) (see Figure 2). The initial model that we used is very simple in order to keep a low complexity when we have many (e.g., 1000) cores: each core has an associated temperature value and the DTS considers also that value when deciding where to schedule threads (the precise algorithm will be detailed later on in the project).

3.1 State of the art and related work

In order to have an appropriate view of the state of the art, we analyzed some related work on Dynamic Thermal Modeling and in particular in multicores and on Temperature-aware thread scheduling and allocation.

Dynamic Thermal Modeling (DTM)

Dynamic Thermal Modeling has been studied in several works [Dhodapkar00], [Brooks01], [Stan03], [Skedron04], [Han06], [Murali07], [Kumar08], [Mutapcic09].

One important observation is that optimizing the temperature by adjusting core speeds and using sophisticated control laws is justified only when the number of cores is greater than **tens** [Mutapcic09]. In TERAFLUX, we aim to manage the temperature of **a thousand** or more cores, so it's relevant to spend some resource of the system even at some cost.

Skadron et al. [Stan03], [Skadron04] introduced 'HotSpot' a framework, which allows for detailed thermal modeling that **can be included in other simulators**. It utilizes an equivalent circuit comprising thermal characteristics of the package and applies 4th order Runge-Kutta methods, with some substantial overhead in calculation. Other simulators that used thermal models are TEMPEST [Dhodapkar00], which is based on SimpleScalar and a Wattch extension that uses power as a proxy to estimate temperature [Brooks01]. This suggest some basic model for temperature that can be considered in TERAFLUX too.

Han et al. [Han06] present an algorithm to estimate and make projection on temperature based on hardware performance counters, but which is much faster than other methods such as the one available in the HotSpot simulator [Stan03]. The work of Kumar et al. [Kumar08] proposed a regression- based thermal model that uses hardware performance counters available in the processor similarly to [Han06]. Both works [Han06], [Kumar08] are only limited to the uniprocessor case, but indicates an opportunity to establish temperature control **even without temperature sensors**, which is very useful for the simulation. This gives the option to avoid using the concept of “temperature sensor” in TERAFLUX.

Murali et al. [Murali07] study an MPSoC consisting of 15 cores (and same number of local memories) and propose a design-time solution of DVFS scheduling based on convex optimization that optimizes performance (total instruction executed) while meeting the given power and temperature constraints. This results in a simple formula that says that **the average frequency of operation of each core should be *w/(n+l)*** where *w* is the total number of cycles of the workload, *n* the number of cores and *l* the latency to finish the computation in each core. This formula could be considered in TERAFLUX models for reference.

Temperature-aware threads scheduling on Multi-cores

Temperature-aware thread scheduling is also used in multi-core systems to optimize the resources performance and to reduce thermal hotspots by distributing heat generation more homogeneously across the silicon chip layout [Powell04], [Annavaram04], [Choi07], [Coskun08], [Kang11].

Powell et al. [Powell04] is the first work that proposes a combination of thread scheduling on a Chip Multiprocessor (CMP) and on Simultaneous Multi-Threaded cores (SMT) in order to avoid overheating cores (and therefore critical temperatures) and try reduce power-density as a consequence. In TERAFLUX, similarly, the DTS accounts for the temperature and may avoid overheating cores in the operation of thread assignment.

Annavaram et al. [Annavaram04] used a 4-way multicore to show that code that presents a large level of thread parallelism can provide an important acceleration when running on a low frequency, lower performance multi-core than a single high-frequency complex architecture core using exactly the same amount of power consumption. The approach that we use in our DTS of distributing as many DF-threads in parallel is exactly moving in the direction of reducing the Energy expended Per Instruction (EPI) based on the available parallelism: DF-threads are mapped on lower-power “Auxiliary Cores” (AC) (cf. D7.1, D7.2).

Choi et al. [Choi07] used an OS-level temperature-aware scheduling, which samples the 24 sensors of a dual-core POWER5 (1.2 GHz) every 4ms (the scheduler tick). Their conclusions are relevant to our work: i**) they confirm that on-chip temperature is closely related to unit utilization and ii) the changes in temperature are in the order of hundreds of ms, which is about two order of magnitude larger of the scheduler tick**. This will permit us to direct our effort to a proper consideration of the temperature variations on the scheduling decisions.

Coskun et al. [Coskun08] proposed to use a combination of off-line (based on Integer Linear Programming) and on-line techniques to minimize temperature hotspots and gradients of the MPSoC with per-core DVFS capability. Chantem et al. [Chantem08] presented a thread assignment and scheduling technique for hard real-time applications in MPSoCs, which uses a mixed integer linear programming solver to minimize the peak temperature of multi-core systems based on the steady-state thermal analysis. They also proposed two heuristic approaches for thread scheduling based on steady-state and transient thermal analysis. These techniques could provide indication on how to distribute computations in order to avoid thermal hotspots. However, all these solutions are design–time approaches.

Kang et al. [Kang11] propose to use a **thread migration** method at runtime to respect power and temperature constraint. This paper is also the first, as of our knowledge to consider such problem in the context of 3D multi-core architectures. In TERAFLUX we use DTS to dynamically schedule task; the migration is unnecessary since the needs are determined before the scheduling of the threads. In other words, TERAFLUX DTS relies on thread assignment rather than thread scheduling.

Temperature/Thermal management

Ebi et al. [Ebi09] propose an proactive power distribution method (named "TAPE") based on a per-core agent implemented in a mixed hardware and software fashion to evenly distribute power and to reduce the peak temperature while meeting the given deadline constraint. The per-core agent performs dynamic voltage and frequency scaling (DVFS) and/or power gating according to measured temperature of the core. Their approach aims to reduce the peak temperature and improves the state-of-the-art techniques like HRTM [Powell04] and PDTM [Yang09]. They use a 96-core platform using MiBench.

In [Kumar08], while the DTM is done through performance counters, as explained above, the temperature management technique (called *HybDTM*) controls the chip temperature by clock-gating or limiting tasks' execution when the estimated temperature reaches a given threshold. In TERAFLUX, we aim to use a similar interaction between policies set at software level (OS, etc.) and lower level estimation and control through the DTS.

In [Zhang08], the authors presented the first stochastic temperature-aware DVFS method to keep the expected latency within the designer specified level, while meeting the condition that the probability of peak temperature exceeding a given threshold is sufficiently small. Their technique is an off-line method while in TERAFLUX we aim at an on-line methodology (which is also in the future work of the same authors).

Donald et al. [Donald06] suggested a two-loop system for an OS based thread migration based on information provided at core level. In their approach, both hardware fine-grained adjustments to cope with thermal emergencies and software based migration for heat balancing and optimized performance is used. A similar approach is supported in the TERAFLUX architecture, but we are also considering power and faults.

The work from Zhu et al. [Zhu 08] stresses the importance of thermal management in 3-D Chip Multiprocessors: the power densities of such chips will require frequent invocations of thermal control. The authors develop a mathematical formulation to balance the workload, the power and the temperature and implement it on a unified hardware and OS framework (named ThermOS).

3.2 Implementation of Thermal model in the Simulator

The Thermal Model in the simulator is currently work in progress, based on our finding on literature. We also believe that an interface very similar to that one designed for power should be used.

3.3 Definition of the granularity

Recent high-performance processors like the Intel Sandy-Bridge contain as many as 12 temperature sensors [Rotem11] and there is appropriate (local) thermal management inside the processor itself. In the TERAFLUX project, we investigate the thermal management of a chip encompassing 1000+ core and therefore focus on what to do at overall level, without necessarily interfering with the per-core thermal management.

The work by Powell et al. [Powell04] also investigates the spatial and temporal granularity to react to overheating (hotspots): if the granularity is too coarse, we may lose opportunities to adaptation. In TERAFLUX, we are targeting system with 1000+ of more cores; therefore **the granularity of a core appears appropriat**e (more local techniques may be applied as well, if needed). For the temporal granularity, it has been observed [Powell04] that it is not a relevant choice as long as we have the ability to stop the computation once a certain threshold value for temperature is reached. Again, this is perfectly in line with the approach that we use in the DTS.

Therefore we choose a single number per-core to characterize the core (average) temperature and this is the granularity level we aim to manage. As explained in this document and in other deliverables (D6.2), our view is that the Distributed Thread Scheduler takes care of an appropriate thread allocation based on the global information it has, including the temperature.

As explained below, this document introduces the general idea of how this can be done in the simulator.

3.4 Model of Node activity to estimate Temperature

The temperature estimation and management framework is still in development. We assume that we can estimate temperature using approaches similar to those found in the literature [Dhodapkar00], [Brooks01], [Stan03], [Skedron04], [Han06], [Murali07], [Kumar08], [Mutapcic09], i.e., by an indirect calculation during the simulation. This calculation has not to be done very frequently, in the order of every hundreds of ms as proposed in other works [Choi07].

3.4.1 Temperature Management in TERAFLUX/COTSon

The Distributed Thread Scheduler (described in D6.2) bookkeeps information about continuations (a tuple) associated to every thread. This tuple contains at least the following information:

- The Synchronization Count (SC), which specifies the number of producer inputs to that thread before it becomes ready to execute.
- The Instruction Pointer (IP) is the pointer to the first instruction of the associated thread in the code memory.
- The Frame Pointer (FP) is the address of the frame in the frame memory allocated for the associated thread.
- The Core Identifier (CID), which identifies the core where the thread is executed.

Another table will provide information about the cores, where each core entry will have the following fields (as shown in Figure 2):

- A POWER indicator, through which the power of the core is known to the DTS;
- A TEMP indicator, through which the temperature of the core is known to the DTS;
- A FAULTS indicator, through which the faultiness of the core is known to the DTS.

For completeness, we recall that other information is stored in the continuation, like the pointers to the eventually allocated memories like Transactional Memory (TM), Owner Writeable memory (OWM) and Thread Local Storage (TLS), already discussed in D7.1, D7.2, D6.1, D6.2 (see Figure 2).

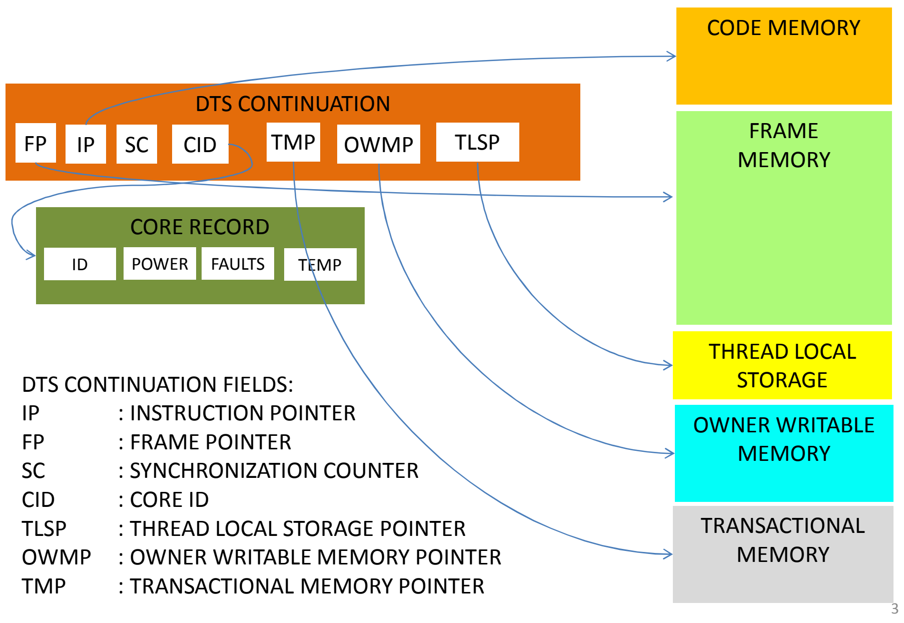

Figure 2 – High Level Model Implementation (a design option for the DTS continuation).

Essentially in TERAFLUX/COTSon we are interested to model the temperature in such a way that the number "TEMP" gets periodically updates. Then the DTS can take into account such number and implement one of the state-of-the-art policies described in Section 3.1, or more sophisticated policies.

The main goal at the moment is to show that we can have a unified management that is able to manage the information in a distributed manner on the chip.

4 Fault-injection support (UAU)

This section reports how the information supplied by the FDU may be used by the DTS to decide how to distribute the threads among the cores, and if possible, also to set the speed of each individual core (e.g., we may decide to reduce the speed of a core that has many faults or high power consumption or temperature before deciding to shut it down).

In Section 3.6 of the previous deliverable D7.2 we explained the implementation of the Fault Detection Unit (FDU) and its interfaces (partner UAU). The main work of UAU with respect to WP7 concerns: (i) the implementation of the FDU and its interfaces (part of Task 7.2), and (ii) the fault injection and fault tracking (part of Task 7.3).

To model the Fault Detection and Recovery approach developed in WP5, UAU is integrating a fault model, which has been initially described in Deliverable D5.1, in the TERAFLUX platform. Therefore, we base the necessary fault injection techniques on the given TERAFLUX architecture, which is composed of a certain number of nodes, which itself consist of cores, a “Node Manager” (D-FDU and D-TSU) and an intra-node interconnect. Among the nodes we assume a 2D-mesh structured interconnection network.

Following WP5’s high level fault model, described in D5.1 and D5.2, we distinguish between core faults and interconnection faults. For core faults, we further distinguish between total faults, partial faults, and retarding faults. For interconnect faults, we differentiate between total faults and partial faults[1].

[1] Please refer to D5.1 for a detailed description of this fault model.

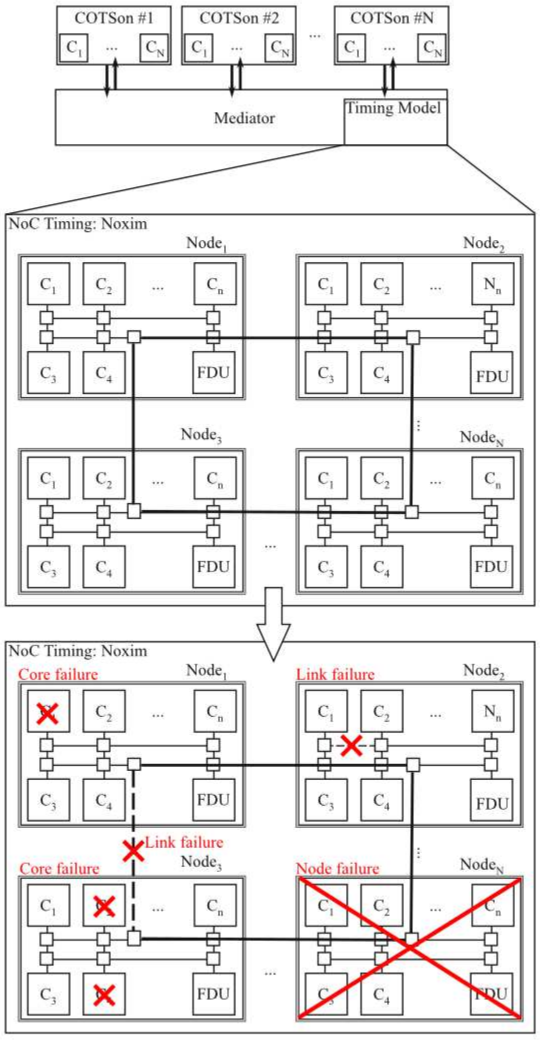

Figure 3 - Faults in COTSon/Noxim.

4.1 Baseline Machine Integration

At this stage of the project UAU has modeled all faults within the Network on Chip Simulator Noxim. Noxim is integrated in the COTSon Mediator for the Network on Chip Timing Model between COTSon nodes (see D7.2). Within Noxim, we model an abstract TERAFLUX fault detection architecture, depicted in Figure 3. This means, Noxim reproduces the TERAFLUX architecture simulated in the COTSon nodes, consisting of nodes, which enclose cores and the D-FDU (see D5.1 and D5.2 for further details). In the next two years UAU will move the current D-FDU implementation, currently running in Noxim as a processing element, onto the COTSon nodes and show that the developed fault detection architecture is capable not only to detect the modeled faults in COTSon but also recover from them.

In the following, we describe how we model faults:

4.2 Core Faults

Total faults are introduced into a single core by stopping the core from sending heartbeats and execution of the current thread.

Partial faults are generated by setting a corresponding Machine Check Architecture register within the core.

Retarding faults are introduced by lowering the IPC of the core (the core runs slower) and by setting the corresponding Performance Counter Registers of the core.

4.2.1 Inter-core Faults

For faults between cores we differentiate between faults in the intra-node interconnect and faults in the inter-node interconnect.

For the intra-node interconnect we model **total faults** by stopping the core from sending heartbeat messages (the core is no longer reachable by a broken link) and **partial faults** by deferring heartbeat messages (the path has become slower by an emulated re-routing).

Since we use the network simulator Noxim for the inter-node network timing we can introduce more elaborated faults, comprising link and router faults, which finally result either in partial faults or total faults. Different flags at a certain routers or links in Noxim can be used for modeling these faults. These flags may be removed later on if the fault is a transient or an intermittent fault.

References

[Annavaram05] M. Annavaram, E. Grochowski, and J. Shen, "Mitigating Amdahl's low through EPI throttling," in Proc. ISCA, Jun. 2005, pp. 298–30

[Bergamaschi08] R. Bergamaschi, H. Guoling, A. Buyuktosunoglu, H. Patel, I. Nair, G. Dittmann, G. Janssen, N. Dhanwada, H. Zhigang, P. Bose, and J. Darringer, "Exploring power management in multicore systems," in Proc. ASPDAC, Mar. 2008, pp. 708–713.

[Brooks02] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors" in Proc. of High-Performance Computer Architecture, (HPCA), Jan. 2001, pp. 171 -182

[Chantem08] T. Chantem, R. P. Dick, and X. S. Hu, "Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs," in Proc. DATE, Mar. 2008, pp. 288–293.

[Chiou09] Derek Chiou, Hari Angepat, Nikhil A. Patil, and Dam Sunwoo, "Accurate Functional-First Multicore Simulators", IEEE COMPUTER ARCHITECTURE LETTERS, VOL. 8, NO. 2, JULY-DECEMBER 2009.

[Choi07] J. Choi, C. Chen-Yong, H. Franke, H. Hamann, A. Weger, and P. Bose, "Thermal-aware task scheduling at the system software level," in Proc. ISLPED, Aug. 2007, pp. 213–218.

[Coskun09] A. K. Coskun, J. L.Ayala, D. Atienza, T. S. Rosing, and Y. Leblebici, "Dynamic thermal management in 3-D multicore architectures," in Proc. DATE, 2009, pp. 1410–1415.

[Coskun08] A. K. Coskun, T. T. Rosing, K. A. Whisnant, and K. C. Gross, "Static and dynamic temperature-aware scheduling for multiprocessor SoCs," IEEE Trans. Very Large Scale Integr. Syst., vol. 16, no. 9, pp. 1127–1140, Sep. 2008.

[COTSon11] http://cotson.svn.sourceforge.net/viewvc/cotson/trunk/src/mcpat/

[Dhodapkar00] A. Dhodapkar, C.H. Lim, G. Cai, and W. R. Daasch "TEMPEST: A Thermal Enabled Multi- Model Power/Performance Estimator," Proc. Workshop on Power-Aware Computer Systems, 2000.

[Donald06] J. Donald and M. Martonosi, "Techniques for multicore thermal management: Classification and new exploration," in Proc. ISCA, 2006, pp. 78–88.

[Ebi09] T. Ebi, M. A. A. Farugue, and J. Henkel, "TAPE: Thermal-aware agent-based power economy for multi/many-core architectures," in Proc. ICCAD, Nov. 2009, pp. 302–309.

[Han06] Y. Han, I. Koren, and C. M. Krishna, "Temptor: A lightweight runtime temperature monitoring tool using performance counters," in Proc. 3rd Workshop TACS, Held Conjunct. ISCA-33, 2006.

[Isci06] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An analysis of efficient multicore global power management policies: Maximizing performance for a given power budget," in Proc. Int. Symp. Microarchitect., Dec. 2006, pp. 347–358.

[Kang11] Kyungsu Kang, Jungsoo Kim, Sungjoo Yoo, Member, and Chong-Min Kyung, "Runtime Power Management of 3-D Multi-Core Architectures Under Peak Power and Temperature Constraints ", IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. 30, NO. 6, JUNE 2011

[Kumar08] A. Kumar, S. Li, P. Li-Shiuan, and N. K. Jha, "System-level dynamic thermal management for high-performance microprocessors," IEEE Trans. Comput. Aided Des. Integr. Circuits Syst., vol. 27, no. 1, pp. 96–108, Jan. 2008.

[Li09] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, Norman P. Jouppi , "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures", ACM MICRO 41, 2009.

[Murali07] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, and G. De Micheli, "Temperature-aware processor frequency assignment for MPSoCs using convex optimization," in Proc. CODES+ISSS, Sep. 2007, pp. 111–116.

[Mutapcic09] A. Mutapcic, S. Boyd, S. Murali, D. Atienza, G. De Micheli, and R. Gupta, "Processor speed control with thermal constraints," IEEE Trans. Circuits Syst. Part I: Reg. Papers, vol. 56, no. 9, pp. 1994–2008, Sep. 2009.

[Naveh06] Alon Naveh, Efrain Rotem, Avi Mendelson, Simcha Gochman, Rahshree Chabukswar, Karthik Krishnan, and Arun Kumar, "Power and Thermal management in the Intel Core Duo Processor", Intel Technology Journal. Vol. 10, Issue 2, May 2006.

[Oh10] D. Oh, N. S. Kim, C.C.P. Chen, A. Davoodi, Y. H. Hu, "Runtime temperature-based power estimation for optimizing throughput of thermal-constrained multi-core processors", Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific, Issue Date: 18-21 Jan. 2010.

[Powell04] M. D. Powell, M. Gomaa, and T. N. Vijaykumar, "Heat-and-Run: Leveraging SMT and CMP to manage power density through the operating system," in Proc. ASPLOS, Nov. 2004, pp. 260–270.

[Rotem11] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, E. Weissmann "2nd Generation Intel* Core* Processor Family: Intel Core i7, i5 and i3", HotChips 2011, Stanford, CA, USA, Aug, 2011.

[Skadron04] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," ACM Trans. Architect. Code Optimization, vol. 1, no. 1, pp. 94–125, Mar. 2004.

[Stan03] M. R. Stan, K. Skadron, M. Barcella, W. Huang, K. Sankaranarayanan, and S. Velusamy. "HotSpot: A Dynamic Compact Thermal Model at the Processor-Architecture Level." Microelectronics Journal: Circuits and Systems, Elsevier, 34(12):1153-65, Dec. 2003.

[Sun07] C. Sun, L. Shang, and R. P. Dick, "3-D multiprocessor system-on-chip thermal optimization," in Proc. Int. Conf. Hardware/Software Codes. Syst. Synthesis, Oct. 2007, pp. 117–122.

[Yang09] Jun Yang, Xiuyi Zhou, Marek Chrobak, Youtao Zhang, and Lingling Jin. 2008. Dynamic Thermal Management through Task Scheduling. In Proceedings of the ISPASS 2008 - IEEE International Symposium on Performance Analysis of Systems and software (ISPASS '08). IEEE Computer Society, Washington, DC, USA, 191-201.

[Zhang08] S. Zhang and K. S. Chatha, "System-level thermal aware design of applications with uncertain execution times," in Proc. ICCAD, Nov. 2008, pp. 242–249.

[Zhao08] G. Zhao, H.-K. Kwan, C.-U. Lei, and N. Wong, "Processor frequency assignment in 3-D MPSoCs under thermal constraints by polynomial programming," in Proc. APCCAS, Nov.–Dec. 2008, pp. 1668–1671.

[Zhou10] X. Zhou, J.Yang,Y. Xu,Y. Zhang, and J. Zhao, "Thermal-aware task scheduling for 3-D multicore processors," IEEE Trans. Parallel Distributed Syst., vol. 21, no. 1, pp. 60–71, Jan. 2010.

[Zhu08] C. Zhu, Z. Gu, L. Shang, R. P. Dick, and R. Joseph, "3-D chip-multiprocessor runtime thermal management," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 27, no. 8, pp. 1479–1492, Aug. 2008.

Appendix 1

COTSon input file (`example_power.in`)

```
options = {
            fastforward="100M",
            time_feedback = true,
            heartbeat = { type="sqlite", dbfile=DBFILE, experiment_id=1,
               experiment_description="test run"},
            max_nanos="200M",
            sampler={ type="dynamic", functional="500k", warming="100k",
               simulation="100k",
            maxfunctional=1000, sensitivity="500",
            variable={"cpu.*.other_exceptions", "cpu.*.iocount"}},
}

one_node_script="run_interactive"
-- display=os.getenv("DISPLAY")

simnow.commands=function()
            use_bsd('1p-reset.bsd')
            use_hdd('karmic64.img')
            set_journal()
            send_keyboard('gcc -S -o - -O3 -c -w /home/user/test.i')
end

function build()
            i=0
            while i < disks() do
                        disk=get_disk(i)
                        disk:timer{ name='disk'..i, type="simple_disk" }
                        i=i+1
            end

            i=0
            while i < nics() do
                        nic=get_nic(i)
                        nic:timer{ name='nic'..i, type="simple_nic" }
                        i=i+1
            end
            mem=Memory{ name="main", latency=150  }

    l2=Cache{ name="l2cache", size="512kB",
                        line_size=16, latency=20, num_sets=4, next=mem,
                        write_policy="WB", write_allocate="true" }
    t2=TLB{ name="l2tlb", page_size="4kB",
                        entries=512, latency=80, num_sets=4, next=mem,
                        write_policy="WB", write_allocate="true" }

            i=0
            while i < cpus() do
                cpu=get_cpu(i)
                -- cpu:timer{ name='cpu'..i, type="timer_dep" }
                cpu:timer{ name='cpu'..i, type="timer1" }
        ic=Cache{ name="icache"..i, size="16kB", line_size=16,
                            latency=0, num_sets=2, next=l2,
                            write_policy="WT", write_allocate="false" }
        dc=Cache{ name="dcache"..i, size="16kB", line_size=16,
                            latency=0, num_sets=2, next=l2,
                            write_policy="WT", write_allocate="false" }
        it=TLB{ name="itlb"..i, page_size="4kB", entries=40,
                            latency=0, num_sets=40, next=t2,
                            write_policy="WT", write_allocate="false" }
        dt=TLB{ name="dtlb"..i, page_size="4kB", entries=40,
                            latency=0, num_sets=40, next=t2,
                            write_policy="WT", write_allocate="false" }

                cpu:instruction_cache(ic)
                cpu:data_cache(dc)
                cpu:instruction_tlb(it)
                cpu:data_tlb(dt)
                        i=i+1
    end
end
-- vim:ft=lua
```

McPAT input file

```
<?xml version="1.0" ?>
<!-- experiment test run , cpumodel x86a -->
<component id="root" name="root">
  <component id="system" name="system">
    <param name="homogeneous_cores" value="1"/>
    <param name="homogeneous_L2s" value="1"/>
    <param name="number_of_cores" value="1"/>
    <param name="number_cache_levels" value="2"/>
    <param name="number_of_L2s" value="1"/>
    <param name="Private_L2" value="0"/>
    <param name="number_of_L3s" value="0"/>
    <param name="longer_channel_device" value="1"/>
    <param name="number_of_NoCs" value="1"/>
    <param name="number_of_L2Directories" value="0"/>
    <param name="device_type" value="0"/>
    <param name="number_of_L1Directories" value="0"/>
    <param name="virtual_memory_page_size" value="4096"/>
    <param name="interconnect_projection_type" value="0"/>
    <param name="homogeneous_L2Directorys" value="1"/>
    <param name="homogeneous_L1Directorys" value="1"/>
    <param name="temperature" value="380"/>
    <param name="machine_bits" value="64"/>
    <param name="homogeneous_ccs" value="1"/>
    <param name="virtual_address_width" value="64"/>
    <param name="core_tech_node" value="22"/>
    <param name="homogeneous_L3s" value="1"/>
    <param name="homogeneous_NoCs" value="1"/>
    <param name="physical_address_width" value="52"/>
    <param name="target_core_clockrate" value="3200"/>
    <stat name="total_cycles" value="640300000"/>
    <stat name="idle_cycles" value="636532731"/>
    <stat name="busy_cycles" value="3767269"/>
    <component id="system.core0" name="core0">
      <param name="clock_rate" value="3200"/>
      <param name="phy_Regs_IRF_size" value="256"/>
      <param name="instruction_window_scheme" value="0"/>
      <param name="instruction_length" value="32"/>
      <param name="archi_Regs_FRF_size" value="32"/>
      <param name="pipelines_per_core" value="1"/>
      <param name="number_hardware_threads" value="2"/>
      <param name="number_of_BTB" value="2"/>
      <param name="archi_Regs_IRF_size" value="16"/>
      <param name="machine_type" value="0"/>
      <param name="RAS_size" value="64"/>
      <param name="number_instruction_fetch_ports" value="1"/>
      <param name="1" value="pipeline_depth"/>
      <param name="phy_Regs_FRF_size" value="256"/>
      <param name="x86" value="1"/>
      <param name="opcode_width" value="16"/>
      <param name="FPU_per_core" value="2"/>
      <param name="decoded_stream_buffer_size" value="16"/>
      <param name="MUL_per_core" value="1"/>
      <param name="opt_local" value="0"/>
      <param name="number_of_BPT" value="2"/>
      <param name="instruction_window_size" value="64"/>
      <param name="micro_opcode_width" value="8"/>
      <param name="ROB_size" value="80"/>
      <param name="fp_instruction_window_size" value="64"/>
      <param name="31" value="31"/>
      <param name="instruction_buffer_size" value="32"/>
      <param name="fetch_width" value="1"/>
      <param name="decode_width" value="1"/>
      <param name="issue_width" value="1"/>
      <param name="peak_issue_width" value="1"/>
      <param name="commit_width" value="1"/>
      <param name="fp_issue_width" value="1"/>
      <param name="ALU_per_core" value="1"/>
      <stat name="total_instructions" value="6283310"/>
      <stat name="int_instructions" value="4097794"/>
      <stat name="fp_instructions" value="0"/>
      <stat name="branch_instructions" value="2185516"/>
      <stat name="branch_mispredictions" value="1160432"/>
      <stat name="load_instructions" value="1181316"/>
      <stat name="store_instructions" value="1181316"/>
      <stat name="committed_instructions" value="6283310"/>
      <stat name="committed_int_instructions" value="4097794"/>
      <stat name="committed_fp_instructions" value="0"/>
      <stat name="pipeline_duty_cycle" value="1.61930720397461"/>
      <stat name="ROB_reads" value="6283310"/>
      <stat name="ROB_writes" value="6283310"/>
      <stat name="rename_reads" value="0"/>
      <stat name="rename_writes" value="0"/>
      <stat name="fp_rename_reads" value="0"/>
      <stat name="fp_rename_writes" value="0"/>
      <stat name="inst_window_reads" value="6283310"/>
      <stat name="inst_window_writes" value="6283310"/>
      <stat name="inst_window_wakeup_accesses" value="0"/>
      <stat name="fp_inst_window_reads" value="0"/>
      <stat name="fp_inst_window_writes" value="0"/>
      <stat name="fp_inst_window_wakeup_accesses" value="0"/>
      <stat name="int_regfile_reads" value="0"/>
      <stat name="int_regfile_writes" value="0"/>
      <stat name="float_regfile_reads" value="0"/>
      <stat name="float_regfile_writes" value="0"/>
      <stat name="ialu_accesses" value="4097794"/>
      <stat name="fpu_accesses" value="0"/>
      <stat name="cdb_alu_accesses" value="0"/>
      <stat name="cdb_fpu_accesses" value="0"/>
      <stat name="MemManU_D_duty_cycle" value="0.5"/>
      <stat name="FPU_cdb_duty_cycle" value="0.3"/>
      <stat name="MUL_duty_cycle" value="0.3"/>
      <stat name="MUL_cdb_duty_cycle" value="0.3"/>
      <stat name="LSU_duty_cycle" value="0.5"/>
      <stat name="IFU_duty_cycle" value="1"/>
      <stat name="MemManU_I_duty_cycle" value="1"/>
      <stat name="ALU_cdb_duty_cycle" value="1"/>
      <stat name="ALU_duty_cycle" value="1"/>
      <stat name="FPU_duty_cycle" value="0.3"/>
      <component id="system.core0.predictor" name="PBT">
        <param name="local_predictor_size" value="14,1"/>
        <param name="local_predictor_entries" value="16384"/>
        <param name="global_predictor_entries" value="16384"/>
        <param name="global_predictor_bits" value="1"/>
        <param name="chooser_predictor_entries" value="16384"/>
        <param name="chooser_predictor_bits" value="1"/>
        <param name="load_predictor" value="14,1,16384"/>
        <param name="global_predictor" value="16384,1"/>
        <param name="predictor_chooser" value="16384,1"/>
      </component> <!-- PBT -->
      <component id="system.core0.itlb" name="itlb">
        <param name="number_entries" value="40"/>
        <stat name="total_accesses" value="3552384"/>
        <stat name="total_misses" value="0"/>
        <stat name="conflicts" value="0"/>
      </component> <!-- itlb -->
      <component id="system.core0.icache" name="icache">
```

```
        <param name="icache_config" value="16384,16,2,1,1,0,16,1"/>
        <param name="buffer_sizes" value="16, 16, 16, 0"/>
        <stat name="read_accesses" value="3552384"/>
        <stat name="read_misses" value="1874"/>
        <stat name="write_accesses" value="0"/>
        <stat name="write_misses" value="0"/>
        <stat name="conflicts" value="0"/>
      </component> <!-- icache -->
      <component id="system.core0.dtlb" name="dtlb">
        <param name="number_entries" value="40"/>
        <stat name="total_accesses" value="2356877"/>
        <stat name="total_misses" value="0"/>
        <stat name="conflicts" value="0"/>
      </component> <!-- dtlb -->
      <component id="system.core0.dcache" name="dcache">
        <param name="dcache_config" value="16384,16,2,1,1,0,16,0"/>
        <param name="buffer_sizes" value="16, 16, 16, 1"/>
        <stat name="read_accesses" value="1181316"/>
        <stat name="read_misses" value="0"/>
        <stat name="write_accesses" value="1175560"/>
        <stat name="write_misses" value="0"/>
        <stat name="conflicts" value="0"/>
      </component> <!-- dcache -->
      <component id="system.core0.BTB" name="BTB">
        <param name="BTB_config" value="5120,4,2,1, 1,3"/>
        <stat name="read_accesses" value="2185516"/>
        <stat name="write_accesses" value="0"/>
      </component> <!-- BTB -->
    </component> <!-- cpu0 -->
    <component id="system.L1Directory0" name="L1Directory0">
      <param name="ports" value="1,1,1"/>
      <param name="device_type" value="0"/>
      <param name="buffer_sizes" value="8,8,8,8"/>
      <param name="Dir_config" value="4096,2,0,1,100,100,8"/>
      <param name="Directory_type" value="0"/>
      <stat name="read_accesses" value="0"/>
      <stat name="write_accesses" value="0"/>
      <stat name="read_misses" value="0"/>
      <stat name="write_misses" value="0"/>
      <stat name="conflicts" value="0"/>
    </component> <!-- L1Directory0 -->
    <component id="system.L2Directory0" name="L2Directory0">
      <param name="ports" value="1,1,1"/>
      <param name="device_type" value="0"/>
      <param name="buffer_sizes" value="8,8,8,8"/>
      <param name="Dir_config" value="4096,2,0,1,100,100,8"/>
      <param name="Directory_type" value="0"/>
      <stat name="read_accesses" value="0"/>
      <stat name="write_accesses" value="0"/>
      <stat name="read_misses" value="0"/>
      <stat name="write_misses" value="0"/>
      <stat name="conflicts" value="0"/>
    </component> <!-- L2Directory0 -->
    <component id="system.L20" name="L20">
      <param name="L2_config" value="524288,16,4,1,1,20,16,0"/>
      <param name="buffer_sizes" value="16, 16, 16, 16"/>
      <param name="clockrate" value="3200"/>
      <param name="ports" value="1,1,1"/>
      <param name="device_type" value="0"/>
      <stat name="read_accesses" value="1874"/>
      <stat name="read_misses" value="1874"/>
      <stat name="write_accesses" value="1175560"/>
      <stat name="write_misses" value="0"/>
      <stat name="conflicts" value="0"/>
    </component> <!-- L20 -->
    <component id="system.L30" name="L30">
      <param name="L3_config" value=",,,1,1,,,0"/>
      <param name="buffer_sizes" value="16, 16, 16, 1"/>
      <stat name="read_accesses" value="0"/>
      <stat name="read_misses" value="0"/>
      <stat name="write_accesses" value="0"/>
      <stat name="write_misses" value="0"/>
      <stat name="conflicts" value="0"/>
    </component> <!-- L30 -->
    <component id="system.Noc0" name="noc0">
      <param name="flit_bits" value="256"/>
      <param name="horizontal_nodes" value="1"/>
      <param name="has_global_link" value="0"/>
      <param name="link_throughput" value="1"/>
      <param name="link_routing_over_percentage" value="0.5"/>
      <param name="chip_coverage" value="1"/>
      <param name="vertical_nodes" value="1"/>
      <param name="input_ports" value="1"/>
      <param name="type" value="0"/>
      <param name="output_ports" value="1"/>
      <param name="clockrate" value="3200"/>
      <stat name="total_accesses" value="0"/>
      <stat name="duty_cycle" value="1"/>
    </component> <!-- noc0 -->
    <component id="system.mem" name="mem">
      <param name="mem_tech_node" value="32"/>
      <param name="capacity_per_channel" value="4096"/>
      <param name="burstlength_of_DRAM_chip" value="8"/>
      <param name="page_size_of_DRAM_chip" value="8"/>
      <param name="number_ranks" value="2"/>
      <param name="output_width_of_DRAM_chip" value="8"/>
      <param name="internal_prefetch_of_DRAM_chip" value="4"/>
      <param name="Block_width_of_DRAM_chip" value="64"/>
      <param name="device_clock" value="200"/>
      <param name="num_banks_of_DRAM_chip" value="8"/>
      <param name="peak_transfer_rate" value="6400"/>
      <stat name="memory_accesses" value="1874"/>
    </component> <!-- mem -->
    <component id="system.mc" name="mc">
      <param name="IO_buffer_size_per_channel" value="32"/>
      <param name="req_window_size_per_channel" value="32"/>
      <param name="llc_line_length" value="64"/>
      <param name="mc_clock" value="400"/>
      <param name="number_ranks" value="2"/>
      <param name="addressbus_width" value="51"/>
      <param name="memory_channels_per_mc" value="1"/>
      <param name="number_mcs" value="0"/>
      <param name="peak_transfer_rate" value="6400"/>
      <param name="databus_width" value="128"/>
      <stat name="memory_accesses" value="1874"/>
      <stat name="memory_reads" value="1874"/>
      <stat name="memory_writes" value="0"/>
    </component> <!-- mc -->
    <component id="system.niu" name="niu">
      <param name="number_units" value="1"/>
      <param name="clockrate" value="350"/>
      <param name="type" value="0"/>
      <stat name="duty_cycle" value="1.0"/>
      <stat name="total_load_perc" value="0"/>
    </component> <!-- niu -->
    <component id="system.pcie" name="pcie">
      <param name="number_units" value="1"/>
      <param name="withPHY" value="1"/>
      <param name="num_channels" value="8"/>
      <param name="clockrate" value="350"/>
      <param name="type" value="0"/>
      <stat name="duty_cycle" value="1.0"/>
      <stat name="total_load_perc" value="0"/>
    </component> <!-- pcie -->
    <component id="system.flashc" name="flashc">
      <param name="number_flashcs" value="4"/>
      <param name="withPHY" value="1"/>
      <param name="type" value="1"/>
      <param name="peak_transfer_rate" value="200"/>
      <stat name="duty_cycle" value="1.0"/>
      <stat name="total_load_perc" value="0"/>
    </component> <!-- flashc -->
  </component> <!-- system -->
</component> <!-- root -->
```

McPAT output results

```
McPAT (version 0.8 of Aug, 2010) is computing the target processor...
McPAT (version 0.8 of Aug, 2010) results  (current print level is 5)
*********************************************************************
Technology 22 nm
  Using Long Channel Devices When Appropriate
  Interconnect metal projection= aggressive interconnect technology
projection
  Core clock Rate(MHz) 3200

*********************************************************************
Processor:
  Area = 5.71035 mm^2
  Peak Power = 3.26812 W
  Total Leakage = 1.07086 W
  Peak Dynamic = 2.19727 W
  Subthreshold Leakage = 1.06684 W
  Gate Leakage = 0.00401736 W
  Runtime Dynamic = 0.26991 W

  Total Cores: 1 cores
  Device Type= ITRS high performance device type
    Area = 1.80503 mm^2
    Peak Dynamic = 0.84629 W
    Subthreshold Leakage = 0.593165 W
    Gate Leakage = 0.00126014 W
    Runtime Dynamic = 0.26966 W

  Total L2s:
  Device Type= ITRS high performance device type
    Area = 0.937911 mm^2
    Peak Dynamic = 0.165416 W
    Subthreshold Leakage = 0.217494 W
    Gate Leakage = 0.000684481 W
    Runtime Dynamic = 0.000249549 W

  Total NoCs (Network/Bus):
  Device Type= ITRS high performance device type
    Area = 0.0611096 mm^2
    Peak Dynamic = 0.643255 W
    Subthreshold Leakage = 0.0179458 W
    Gate Leakage = 5.82246e-05 W
    Runtime Dynamic = 0 W

  Total Flash/SSD Controllers: 4 Flash/SSD Controllers
  Device Type= ITRS high performance device type
    Area = 0.131969 mm^2
    Peak Dynamic = 0.0964636 W
    Subthreshold Leakage = 0.0165631 W
    Gate Leakage = 0.000140058 W
    Runtime Dynamic = 0 W

  Total NIUs: 1 Network Interface Units
  Device Type= ITRS high performance device type
    Area = 1.30046 mm^2
    Peak Dynamic = 0.176111 W
    Subthreshold Leakage = 0.100884 W
    Gate Leakage = 0.000853082 W
    Runtime Dynamic = 0 W

  Total PCIes: 1 PCIe Controllers
  Device Type= ITRS high performance device type
    Area = 1.47387 mm^2
    Peak Dynamic = 0.26973 W
    Subthreshold Leakage = 0.120787 W
    Gate Leakage = 0.00102138 W
    Runtime Dynamic = 0 W

*********************************************************************
Core:
      Area = 1.80503 mm^2
      Peak Dynamic = 0.84629 W
      Subthreshold Leakage = 0.593165 W
      Gate Leakage = 0.00126014 W
      Runtime Dynamic = 0.26966 W
      Instruction Fetch Unit:
        Area = 0.195882 mm^2
        Peak Dynamic = 0.110828 W
        Subthreshold Leakage = 0.204435 W
        Gate Leakage = 0.000361928 W
        Runtime Dynamic = 0.000602019 W

          Instruction Cache:
            Area = 0.085274 mm^2
            Peak Dynamic = 0.0492197 W
            Subthreshold Leakage = 0.0165821 W
            Gate Leakage = 2.46382e-05 W
            Runtime Dynamic = 0.000134032 W

          Instruction Buffer:
            Area = 0.000754971 mm^2
            Peak Dynamic = 0.00217751 W
            Subthreshold Leakage = 0.000209095 W
            Gate Leakage = 3.27157e-07 W
            Runtime Dynamic = 2.13681e-05 W

          Instruction Decoder:
            Area = 0.106826 mm^2
            Peak Dynamic = 0.0455126 W
            Subthreshold Leakage = 0.184982 W
            Gate Leakage = 0.000321652 W
            Runtime Dynamic = 0.000446619 W

      Renaming Unit:
        Area = 0.0469804 mm^2
        Peak Dynamic = 0.0306179 W
        Subthreshold Leakage = 0.00835367 W
        Gate Leakage = 2.32158e-05 W
        Runtime Dynamic = 2.17373e-11 W

          Int Front End RAT:
            Area = 0.0143549 mm^2
            Peak Dynamic = 0.00515925 W
            Subthreshold Leakage = 0.00151142 W
            Gate Leakage = 2.10606e-06 W
            Runtime Dynamic = 0 W

          FP Front End RAT:
            Area = 0.0199486 mm^2
            Peak Dynamic = 0.00673318 W
            Subthreshold Leakage = 0.00273252 W
            Gate Leakage = 3.84272e-06 W
            Runtime Dynamic = 0 W

          Free List:
            Area = 0.00417332 mm^2
            Peak Dynamic = 0.00167418 W
            Subthreshold Leakage = 0.000603379 W
            Gate Leakage = 7.82638e-07 W
            Runtime Dynamic = 0 W

          Int Retire RAT:
            Area = 0.000651412 mm^2
            Peak Dynamic = 0.000729371 W
            Subthreshold Leakage = 0.000120592 W
            Gate Leakage = 1.9569e-07 W
            Runtime Dynamic = 0 W

          FP Retire RAT:
            Area = 0.000651412 mm^2
            Peak Dynamic = 0.000729371 W
            Subthreshold Leakage = 0.000120592 W
            Gate Leakage = 1.9569e-07 W
            Runtime Dynamic = 0 W

          FP Free List:
            Area = 0.00417332 mm^2
            Peak Dynamic = 0.00167418 W
            Subthreshold Leakage = 0.000603379 W
            Gate Leakage = 7.82638e-07 W
            Runtime Dynamic = 0 W

      Load Store Unit:
        Area = 0.140683 mm^2
        Peak Dynamic = 0.0516465 W
        Subthreshold Leakage = 0.0205757 W
        Gate Leakage = 4.31955e-05 W
        Runtime Dynamic = 0.00017223 W

          Data Cache:
            Area = 0.085274 mm^2
            Peak Dynamic = 0.0372062 W
            Subthreshold Leakage = 0.0165821 W
            Gate Leakage = 2.46382e-05 W
            Runtime Dynamic = 8.94167e-05 W

          LoadQ:
            Area = 0.014924 mm^2
            Peak Dynamic = 0.00374056 W
            Subthreshold Leakage = 0.000665865 W
            Gate Leakage = 1.62348e-06 W
            Runtime Dynamic = 2.76045e-05 W

          StoreQ:
            Area = 0.014924 mm^2
            Peak Dynamic = 0.00374056 W
            Subthreshold Leakage = 0.000665865 W
            Gate Leakage = 1.62348e-06 W
            Runtime Dynamic = 5.52089e-05 W

      Memory Management Unit:
        Area = 0.0215983 mm^2
        Peak Dynamic = 0.0220847 W
        Subthreshold Leakage = 0.00526948 W
        Gate Leakage = 2.10997e-05 W
        Runtime Dynamic = 7.16525e-05 W

          Itlb:
            Area = 0.00928538 mm^2
            Peak Dynamic = 0.00776393 W
            Subthreshold Leakage = 0.00130384 W
            Gate Leakage = 2.89467e-06 W
            Runtime Dynamic = 4.30743e-05 W

          Dtlb:
            Area = 0.00928538 mm^2
            Peak Dynamic = 0.00388197 W
            Subthreshold Leakage = 0.00130384 W
            Gate Leakage = 2.89467e-06 W
            Runtime Dynamic = 2.85782e-05 W

      Execution Unit:
        Area = 1.39989 mm^2
        Peak Dynamic = 0.631113 W
        Subthreshold Leakage = 0.354532 W
        Gate Leakage = 0.000810696 W
        Runtime Dynamic = 0.268814 W

          Register Files:
            Area = 0.111148 mm^2
            Peak Dynamic = 0.0343728 W
            Subthreshold Leakage = 0.0095046 W
            Gate Leakage = 1.34297e-05 W
            Runtime Dynamic = 0 W

              Integer RF:
                Area = 0.0555739 mm^2
                Peak Dynamic = 0.0280595 W
                Subthreshold Leakage = 0.0047523 W
                Gate Leakage = 6.71487e-06 W
                Runtime Dynamic = 0 W

              Floating Point RF:
                Area = 0.0555739 mm^2
                Peak Dynamic = 0.00631338 W
                Subthreshold Leakage = 0.0047523 W
                Gate Leakage = 6.71487e-06 W
                Runtime Dynamic = 0 W

          Instruction Scheduler:
            Area = 0.0127407 mm^2
            Peak Dynamic = 0.0257902 W
            Subthreshold Leakage = 0.00247094 W
            Gate Leakage = 5.09262e-06 W
            Runtime Dynamic = 0.000106882 W

              Instruction Window:
                Area = 0.00226786 mm^2
                Peak Dynamic = 0.00712748 W
                Subthreshold Leakage = 0.000612285 W
                Gate Leakage = 1.53483e-06 W
                Runtime Dynamic = 3.61554e-05 W

              FP Instruction Window:
                Area = 0.00509946 mm^2
                Peak Dynamic = 0.0114554 W
                Subthreshold Leakage = 0.00110068 W
                Gate Leakage = 2.44905e-06 W
                Runtime Dynamic = 0 W

              ROB:
                Area = 0.00537336 mm^2
                Peak Dynamic = 0.00720734 W
                Subthreshold Leakage = 0.000757978 W
                Gate Leakage = 1.10874e-06 W
                Runtime Dynamic = 7.07262e-05 W

          Integer ALUs (Count: 1 ):
            Area = 0.0384544 mm^2
            Peak Dynamic = 0.140884 W
            Subthreshold Leakage = 0.0279667 W
            Gate Leakage = 6.39207e-05 W
            Runtime Dynamic = 0.0452976 W

          Floating Point Units (FPUs) (Count: 2 ):
            Area = 1.11344 mm^2
            Peak Dynamic = 0.213378 W
            Subthreshold Leakage = 0.202442 W
            Gate Leakage = 0.000462702 W
            Runtime Dynamic = 0.134046 W

          Complex ALUs (Mul/Div) (Count: 1 ):
            Area = 0.115363 mm^2
            Peak Dynamic = 0.0845306 W
            Subthreshold Leakage = 0.0839 W
            Gate Leakage = 0.000191762 W
            Runtime Dynamic = 0.0893639 W

          Results Broadcast Bus:
            Area Overhead = 0.00571706 mm^2
            Peak Dynamic = 0.118238 W
            Subthreshold Leakage = 0.0255858 W
            Gate Leakage = 5.84789e-05 W
            Runtime Dynamic = 8.9801e-11 W

*****************************************************************************
L2
      Area = 0.937911 mm^2
      Peak Dynamic = 0.165416 W
      Subthreshold Leakage = 0.217494 W
      Gate Leakage = 0.000684481 W
      Runtime Dynamic = 0.000249549 W

*****************************************************************************
Flash Controller:
      Area = 0.0329922 mm^2
      Peak Dynamic = 0.0241159 W
      Subthreshold Leakage = 0.00414076 W
      Gate Leakage = 3.50146e-05 W
      Runtime Dynamic = 0 W

*****************************************************************************
NIU:
      Area = 1.30046 mm^2
      Peak Dynamic = 0.176111 W
      Subthreshold Leakage = 0.100884 W
      Gate Leakage = 0.000853082 W
      Runtime Dynamic = 0 W
```

```
*********************************************************************
PCIe:
      Area = 1.47387 mm^2
      Peak Dynamic = 0.26973 W
      Subthreshold Leakage = 0.120787 W
      Gate Leakage = 0.00102138 W
      Runtime Dynamic = 0 W

*********************************************************************
BUSES
      Area = 0.0611096 mm^2
      Peak Dynamic = 0.643255 W
      Subthreshold Leakage = 0.0179458 W
      Gate Leakage = 5.82246e-05 W
      Runtime Dynamic = 0 W

      Bus:
        Area = 0.0611096 mm^2
        Peak Dynamic = 0.643255 W
        Subthreshold Leakage = 0.0179458 W
        Gate Leakage = 5.82246e-05 W
        Runtime Dynamic = 0 W

**********************************************
```