|  |
| --- |
|  |
| CMPexplore |
| User guide |
| Version 0.2.1, 04 Feb 2014 |

|  |
| --- |
|  |

Contents

[I. Introduction 2](#_Toc355713408)

[II. Build and Installation 2](#_Toc355713409)

[III. A Quick Starter: Running an Exploration 3](#_Toc355713410)

[IV. Configuring the Design Space 4](#_Toc355713411)

[V. Workload Specification 6](#_Toc355713412)

[VI. Command Line Arguments 7](#_Toc355713413)

[VII. Exploration Mode 8](#_Toc355713414)

[VIII. Modeling a Particular Configuration 9](#_Toc355713415)

[IX. Conversion Mode 11](#_Toc355713416)

[X. Hidden Parameters (require recompilation) 12](#_Toc355713417)

[XI. History 13](#_Toc355713418)

[XII. Contacts 13](#_Toc355713419)

# Introduction

CMPexplore is a framework for architectural exploration of hierarchical chip multiprocessors (CMPs). Given a description of the architectural design space in terms of the **models** of CMP components (e.g. core architectures, cache sizes, interconnect topologies, etc.) and **constraints** (e.g. area and power) the tool searches for the architectures that maximize the system throughput in instructions-per-cycle, IPC.

The assumptions and details of the models behind the implementation can be found in the following papers:

|  |  |
| --- | --- |
|  | **N. Nikitin, J. de San Pedro and J. Cortadella.**  *"* *Architectural Exploration of Large-Scale Hierarchical Chip Multiprocessors."*  In IEEE TCAD, Oct 2013.  **N. Nikitin, J. de San Pedro, J. Carmona and J. Cortadella.**  *"Analytical performance modeling of hierarchical interconnect fabrics."*  In Proc. The 6th ACM/IEEE International Symposium on Networks-on-Chip (NOCS), May 2012. |

The tool currently supports two-level interconnects with a global mesh at the top-level, and local buses, crossbars, uni- and bi-directional rings at the cluster level.

# Build and Installation

To build the exploration tool just run make in the cmpexplore directory. You should finally see the ./cmpexplore executable.

Note: this release includes the **newmat10D** library, available at [*http://www.robertnz.net/nm10.htm*](http://www.robertnz.net/nm10.htm)

# A Quick Starter: Running an Exploration

Invoke the following command from the current tool directory:

./cmpexplore –print 10 –dump\_configs

This will run an *exhaustive* exploration of the design space given by the configuration file ./config.txt. Full description of the configuration file follows in the next section. As exploration has finished, it prints out 10 best architectures (as given by the –print argument), ordered by the best throughput. The argument -dump\_configs will store the architectures in the test directory. The output will look as follows:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Stats: ReportAllConfigs \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

x= 5; y= 4; P1= 6; L1=0.064; L2= 1; P#=120; L3= 60MB; Ic=bRing; A=342.7; Pwr = 179.1 Thr = 82.783

x= 6; y= 5; P1= 4; L1=0.064; L2= 1; P#=120; L3= 60MB; Ic=bRing; A=345.2; Pwr = 191.5 Thr = 82.618

x= 7; y= 4; P1= 4; L1=0.128; L2= 1; P#=112; L3= 56MB; Ic=bRing; A=329.4; Pwr = 184.3 Thr = 79.528

x= 4; y= 4; P1= 7; L1=0.128; L2= 1; P#=112; L3= 64MB; Ic=bRing; A=334.4; Pwr = 168.8 Thr = 79.273

x= 8; y= 7; P1= 2; L1=0.128; L2= 1; P#=112; L3= 56MB; Ic=bRing; A=336.4; Pwr = 213.6 Thr = 78.648

x= 7; y= 4; P1= 4; L1=0.064; L2= 1; P#=112; L3= 56MB; Ic=bRing; A=322.2; Pwr = 181.1 Thr = 77.611

x= 4; y= 4; P1= 7; L1=0.064; L2= 1; P#=112; L3= 80MB; Ic=bRing; A=343.2; Pwr = 166.7 Thr = 77.385

x= 8; y= 7; P1= 2; L1=0.064; L2= 1; P#=112; L3= 56MB; Ic=bRing; A=329.2; Pwr = 210.3 Thr = 76.748

x= 9; y= 6; P1= 2; L1=0.128; L2= 1; P#=108; L3= 54MB; Ic=bRing; A=324.4; Pwr = 207.5 Thr = 76.099

x= 6; y= 6; P1= 3; L1=0.128; L2= 1; P#=108; L3= 72MB; Ic=bRing; A=337.9; Pwr = 189.6 Thr = 75.899

790 configurations explored

Total time = 28.718

* The **x** and **y** parameters show the dimensions of global mesh.
* The next three parameters define the core (a.k.a. processor) configuration of one cluster: the name and the number of cores *per cluster* is given for each core type (e.g. P1 = 6), followed by the L1 and L2 cache sizes in MB *per core*.
* **P#** shows the total number of cores per chip.
* **L3** is the total amount of L3 cache distributed over the chip.
* **Ic** is the type of cluster interconnect: **Bus-N/xBar-N** is for a bus/crossbar with the latency of N cycles, **uRing** is for a uni-directional ring and **bRing** is for a bi-directional ring.
* The **A**, **Pwr** and **Thr** columns define the architecture metrics, as obtained by the analytical model, namely the area, power and throughput (IPC).

E.g. the best architecture for this exploration example is a 5x4 mesh with 120 cores, distributed among 20 clusters, each consisting of 6 P1 cores with 64KB L1 and 1MB L2. The total amount of L3 cache is 60Mb. The modeled power and throughput of this architecture are about **179 W** and **83 IPC**, respectively.

# Configuring the Design Space

The parameters of the design space for exploration have to be specified in the configuration file, such as config.txt, found in the same directory. The description of every parameter follows.

|  |  |  |
| --- | --- | --- |
| *Parameter name* | *Description* | *Value specification* |
| gMeshDimX | Global mesh X-dimensions | list of integers |
| gMeshDimY | Global mesh Y-dimensions | list of integers |
| clusterIcType | Type of the cluster interconnect | bus, xbar, uring, bring |
| maxArea | The chip area constraint [mm^2] | Real |
| processor  Name  Area  OoO  Freq  Epi  Pleak  l1Size  l2Size | Introduces a processor (core) type with a set of parameters, listed below:  Core type name  Area of the core [mm^2]  In-order (0) vs. out-of-order (1) architecture  Core frequency [GHz]  Energy per instruction [nJ]  Leakage power [W]  Possible L1 cache sizes  Possible L2 cache sizes | string (required keyword)  string  real  integer (0 or 1)  real  real  real  list of reals  list of reals |
| maxProcAreaPerCluster | Maximum fraction of cluster area occupied by the cores (1.0 means all area **can** be used) | real (in the 0.0-1.0 range) |
| numL3PerCluster | Number of L3 cache modules per cluster | list of integers |
| memDensity | Area of 1Mb of the cache memory [mm^2] | Real |
| wlFile | Workload specification file  (see Sect. Workload Specification) | path (string) |
| l3LatencyOfSize | Dependency of the cache latency on cache size | function (see Note 3 below) |
| l3ShareDegreeOfPNum | Dependency of the cache sharing degree (number of cores sharing a cache line on average) on the total number of cores on chip | Function |
| busTimeOfClusterArea | Dependency of the cluster bus latency on the cluster size | Function |
| xBarDelayOfClusterArea | Dependency of the cluster crossbar latency on the cluster size | Function |
| meshLinkDelayOfClusterArea | Dependency of the mesh link latency on the cluster size | Function |
| l3AccessEnergyOfSize | Dependency of the energy of accessing L3 cache on the cache size | Function |
| l3LeakagePowerOfSize | Dependency of the leakage power of L3 cache on the cache size | function |
| memCtrlAccessEnergy | Energy of accessing a memory controller [nJ] | real |
| memCtrlLeakagePower | Leakage power of memory controller [W] | real |
| frequency | Master clock frequency (memory/IC) [GHz] | real |
| linkWidth | Interconnect link width | integer (supported:  128, 256, 512, 1024) |

**Note 1:** we currently measure all cache sizes in megabytes and use real values like 0.064, 0.128 for the sizes of 64Kb and 128Kb. This is an approximation done for simplicity of implementation, yet providing readable values, and will be fixed in the future version.

**Note 2:** to make the parameter specification easier, one can use ranges of **integer** values: notation [m-n] will be internally converted into the sequence of integers {m, m+1, .., n-1, n}. Range notation can be combined with *enumeration*; hence all the specifications below are *equivalent*:

gMeshDimX = 4, 5, 6, 7, 8, 9, 10

gMeshDimX = [4-10]

gMeshDimX = 4, [5-8], 9, 10

**Note 3:** function specification. In some cases we need to specify how one parameter depends on another. When a discrete range of values is to be explored, a table-based specification may be enough. But for the purposes of continuous range exploration (as well as for shorter notations), we introduce *functions*. Consider the following definition:

l3LatencyOfSize = piecewise (0.064 2) (0.128 3) (0.256 4) (0.512 5) \

(1 6) (2 7) (4 8) (8 9) (12 10)

Here the dependency of the cache latency on the cache size is defined as a linear piecewise function, based on the set of 9 points. This definition is equivalent to the following table:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cache size | 64K | 128K | 256K | 512K | 1M | 2M | 4M | 8M | 12M |
| Cache latency | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

The piecewise function performs linear interpolation for the values between the specified points, as well as constant extrapolation out of the specified region. For instance, for the 96K the latency will be 2 + (3-2)/2 = 2.5 cycles (note that it will be further ceiled to 3 cycles in the tool, but this is somewhat independent of the function behavior). For 32K the latency will be 2 cycles, for the 15M the latency will be 10 cycles (constant extrapolation).

Apart from the piecewise function, several other function types are supported:

linear k b – is a linear dependency in the form k\*x + b

sqrt k – is a square root dependency in the form k\*sqrt(x)

powerlaw k a – is a power-law dependency in the form k\*(x^a)

const k – is a function returning constant value, irrespective of x

# Workload Specification

Since version 0.2 the tool is able to perform exploration for multiple workloads and select the best architecture according to the *maximum aggregate throughput*. The latter is defined by default as the average throughput which configuration delivers for the set of given workloads. However, if required, the aggregating function can be redefined in the code: src/stat/StatConfig.cpp, StatConfig::AggWlObjective().

The wlFile parameter in the configuration file is used to point to a file with workload specification. Note that the file has to contain definition of at least one workload application. For every application, the following fields have to be defined, where the workload name always has to be present and go first.

|  |  |  |
| --- | --- | --- |
| *Parameter name* | *Description* | *Value specification* |
| wlName | Full name of the workload application | string (required keyword) |
| wlShortName | Short name (alias) | string |
| missRatioOfMemSize | Dependency of the miss ratio of executed tasks on the cache size | function (see Note 3 in Sect. Configuring the Design Space) |
| IPC | Ideal throughput of every type of processor, defined in the configuration file, when executing this application | Comma-separated list of reals |
| MPI | Fraction of memory references per instruction, characterizing the workload (for compatibility, this parameter also has to be specified for every processor type) | Comma-separated list of reals (in the 0.0-1.0 range) |

As an example, consider wl\_spx\_nmd\_3cpu.txt found in the project directory. This file defines two workload applications, *soplex* and *namd* from *SPEC CPU2006*. Three processor types are assumed for exploration, such as those described in the configuration file conf\_demo\_1.5b.txt.

# Command Line Arguments

|  |  |
| --- | --- |
| Argument | Description |
| -test <filename> | Model a particular \*.cmp configuration given by the filename. See section VII. |
| -config <filename> | Specify configuration of the design space in filename. Default value = ./config.txt |
| -exp\_mode <string> | Exploration mode: exhaustive (“ex”), simulated annealing (“sa”) or extremal optimization (“eo”). Default value = ex. |
| -max\_power <real> | Maximum power constraint. Configuration whose power exceeds max\_power will not appear in the final list. Default value = 10^6 (no power constraint). |
| -print <integer> | Number of configurations to be printed/saved. Default value = 100. |
| -s\_eff <integer> | Effort of the metaheuristic-based search. Default value = 5. |
| -sa\_a <real> | Scaling factor for search by simulated annealing (“alpha”). Default value = 0.995. |
| -eo\_tau <real> | Power-law exponent for search by extremal optimization (“tau”).  Default value = 1.5. |
| -simcc | Simulate a simplified cache coherency protocol, in contrast to request-reply message pairs (when the flag is off). Assumes 3 physical subnetworks for interconnect, one per request, reply and ack messages. |
| -dump\_configs | Save all output configurations to the ./test directory. For every configuration that appears in the position N of the final output list a file ./test/cmp<N>.cmp will be created. These architectures can be further modeled independently using the  -test mode. The total number of saved configurations is set by -print argument. |
| -a2wa | Conversion mode. Does not execute the tool, but converts architectural files to workload+architectural files (compatible with BookSim). |
| -debug <integer> | Debug mode, use levels from 1 to 5. Prints additional information on execution. |

# Exploration Mode

For large design spaces, the *exhaustive* exploration described in Section III is intractable. CMPexplore implements metaheuristic-based search to discover promising configurations within the large design spaces. Two metaheuristics are available: *Simulated Annealing* (-exp\_mode sa) and *Extremal Optimization* (-exp\_mode eo).

Consider the following exploration scenario:

./cmpexplore -config ./conf\_demo\_1.5b.txt -exp\_mode sa -sa\_a 0.995 -s\_eff 1 -max\_power 150 -print 10

This runs an exploration with the design space described by the ./conf\_demo\_1.5b.txt. This design space is large and includes ~1.5\*109 configurations. To make the exploration tractable, we run a search by simulated annealing, specifying the mode and the scaling factor: -exp\_mode sa -sa\_a 0.995. The effort of the search is set to minimum, for the exploration to finish faster: -s\_eff 1. Larger efforts may deliver better results and will take more time for the exploration to finish. We limit the maximum power of configurations to 150W. We also decide to print only 10 best configurations. The search evolves by printing the configurations which improve on the best throughput found so far.

(Time 0.004) x=2; y=2; uring; P1=1; L1=0.064; L2=0.064; P2=0; L1=0.064; L2=0.064; P3=0; L1=0.064; L2=0.064; A=349.516; Pow=21.5184; Thr=1.74407

...

(Time 9.49259) x=4; y=4; uring; P1=2; L1=0.096; L2=0.64; P2=5; L1=0.096; L2=1; P3=1; L1=0.096; L2=1; A=344.784; Pow=137.954; Thr=61.6267

(Time 15.0449) x=5; y=5; bus; P1=0; L1=0.064; L2=0.512; P2=4; L1=0.128; L2=1; P3=0; L1=0.128; L2=1; A=344.075; Pow=122.19; Thr=61.6891

(Time 47.691) x=5; y=5; bus; P1=1; L1=0.064; L2=1; P2=4; L1=0.128; L2=1; P3=0; L1=0.096; L2=1; A=345.675; Pow=127.065; Thr=66.1029

Finished search (no improvement during the last 4356 iterations)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Stats: ReportAllConfigs \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

x= 5; y= 5; P1= 1; L1=0.064; L2= 1; P2= 4; L1=0.128; L2= 1; P3= 0; L1=0.096; L2= 1; P#=125; L3= 50MB; Ic=Bus-3; A=345.7; Pwr = 127.1 Thr = 66.103

x= 7; y= 4; P1= 1; L1=0.064; L2= 1; P2= 2; L1=0.128; L2= 1; P3= 1; L1=0.128; L2= 1; P#=112; L3= 56MB; Ic=Bus-3; A=341.6; Pwr = 127.2 Thr = 64.813

x= 7; y= 4; P1= 1; L1=0.064; L2= 1; P2= 2; L1=0.128; L2= 1; P3= 1; L1=0.096; L2= 1; P#=112; L3= 56MB; Ic=Bus-3; A=340.7; Pwr = 126.8 Thr = 64.5

x= 6; y= 4; P1= 1; L1=0.064; L2= 1; P2= 3; L1=0.128; L2= 1; P3= 1; L1=0.096; L2= 1; P#=120; L3= 48MB; Ic=Bus-3; A=349.1; Pwr = 125.4 Thr = 63.769

x= 5; y= 5; P1= 0; L1=0.064; L2=0.512; P2= 4; L1=0.128; L2= 1; P3= 0; L1=0.128; L2= 1; P#=100; L3= 100MB; Ic=Bus-3; A=344.1; Pwr = 122.2 Thr = 61.689

x= 4; y= 4; P1= 2; L1=0.096; L2= 0.64; P2= 5; L1=0.096; L2= 1; P3= 1; L1=0.096; L2= 1; P#=128; L3= 48MB; Ic=uRing; A=344.8; Pwr = 138 Thr = 61.627

x= 4; y= 4; P1= 2; L1=0.096; L2=0.512; P2= 5; L1=0.096; L2= 1; P3= 1; L1=0.096; L2= 1; P#=128; L3= 48MB; Ic=uRing; A=340.7; Pwr = 137.3 Thr = 61.074

x= 5; y= 3; P1= 1; L1=0.096; L2= 0.64; P2= 7; L1=0.128; L2=0.768; P3= 0; L1=0.096; L2=0.768; P#=120; L3= 90MB; Ic=uRing; A=345.1; Pwr = 135.9 Thr = 60.715

x= 5; y= 5; P1= 0; L1=0.064; L2=0.512; P2= 4; L1=0.096; L2= 1; P3= 0; L1=0.128; L2= 1; P#=100; L3= 100MB; Ic=Bus-3; A=340.9; Pwr = 121.2 Thr = 60.617

x= 5; y= 3; P1= 2; L1=0.096; L2= 0.64; P2= 5; L1=0.128; L2=0.768; P3= 1; L1=0.096; L2= 1; P#=120; L3= 75MB; Ic=uRing; A=338.2; Pwr = 134.7 Thr = 60.143

Best Thr = 66.103

Params: tCur = 61.495

Total time = 158.03

# Modeling a Particular Configuration

To generate and save N configurations with top throughput for the design space in config.txt, run

./cmpexplore -dump\_configs –print <N>

N files will appear in the directory ./test, with the names cmpX.cmp (X = 1 .. N). Each file is a recursive definition of a CMP architecture. Consider the test/cmp1.cmp after running the above command:

# general parameters

PARAM UnitLen=1.0e-3

PARAM MemDensity=1 # memory density

PARAM L3LatencyDef=8 # L3 cache latency, all L3 modules are of the same size

PARAM MemReplySize=3 # number of flits in the reply packet

PARAM LinkWidth=256 # interconnect link width

PARAM NiDelay=3 # delay of the network interface (between local and global IC)

PARAM WlFile=wl\_spx\_nmd\_1cpu.txt # workload file

# definition of Proc0, a processor with specific parameters (used later)

DEFINE Proc0 PROC Type=0 L1Size=0.064 L1Lat=3 L2Size=1 L2Lat=7 L3SizeEff=0.983607 OoO=1 Area=1.25 Freq=1.6 Epi=0.6 Pleak=0.14 L1Eacc=0.08 L1Pleak=0.01 L2Eacc=0.14 L2Pleak=0.08

# 5x4 mesh description with link delay of 4 cycles and router delay of 3 cycles

MESH Col=5 Row=4 LinkDelay=4 RouterDelay=3

# each of the 20 clusters is defined below;

# a cluster has 8 components, a bi-directional ring IC

# with router delay 1 and no link delay

BRING CompCnt=8 LinkDelay=0 RouterDelay=1

# first 6 components of the cluster are processors of type Proc0 (defined above)

Proc0

Proc0

Proc0

Proc0

Proc0

Proc0

# seventh component is a memory (L3) with size 3MB and specified energy parameters

MEM Size=3 Eacc=0.195 Pleak=0.23

# eighth component is a network interface

NI

# next cluster

BRING CompCnt=8 LinkDelay=0 RouterDelay=1

Proc0

Proc0

Proc0

Proc0

Proc0

Proc0

MEM Size=3 Eacc=0.195 Pleak=0.23

NI

...

# last cluster

BRING CompCnt=8 LinkDelay=0 RouterDelay=1

Proc0

Proc0

Proc0

Proc0

Proc0

Proc0

MEM Size=3 Eacc=0.195 Pleak=0.23

NI

# description of memory controllers

MEMCTRL Location=West Latency=100 Eacc=0.13 Pleak=0.1

MEMCTRL Location=North Latency=100 Eacc=0.13 Pleak=0.1

MEMCTRL Location=South Latency=100 Eacc=0.13 Pleak=0.1

MEMCTRL Location=South Latency=100 Eacc=0.13 Pleak=0.1

MEMCTRL Location=East Latency=100 Eacc=0.13 Pleak=0.1

To perform modeling of a configuration described by a \*.cmp file, use -test argument:

./cmpexplore -test <filename>

Consider the following example:

./cmpexplore -test ./test/cmp1.cmp

The output of this command is:

Area = 342.7 mm^2

WlName = spx: Power = 210.302, Lat = 2.81976, Thr = 112.944

WlName = nmd: Power = 147.927, Lat = 8.90345, Thr = 52.6211

Average: Power = 179.114, Lat = 5.8616, Thr = 82.7827

Total time = 0.032001

The area of the configuration goes first. Afterwards, for every workload defined in the wlFile (wl\_spx\_nmd\_1cpu.txt), the power (W), latency (cycles) and throughput (IPC) of the configuration are shown. The second-to-last line shows average metrics for the given workloads. The last line gives the execution time of the tool.

# Conversion Mode

Running the tool for a given test (-test) with –a2wa parameter will not perform modeling, but rather launch a converter from *architectural* description file (A-file) to *workload+architectural* description files (WA-files), which are compatible with BookSim simulator used in the project. Simulation is the primary need for generation of WA-files.

The difference between these two types of files is that while an A-file contains a path to the workload file, a WA-file incorporates the workload information and hence is self-consistent for simulation. Notice that every WA-file can only incorporate information about one workload application. Hence, one WA-file is generated per every workload.

Generation of architectural files was discussed in the Sect. Modeling a Particular Configuration (by means of the -dump\_configs parameter). To obtain WA-files from an architectural file, e.g. ./test/cmp1.cmp, use the following command:

./cmpexplore -test ./test/cmp1.cmp -a2wa

This will produce files cmp1\_1.cmp, cmp1\_2.cmp in the directory ./test/sim. The first file represents the joint definition of the architecture in ./test/cmp1.cmp, and *soplex* workloads. The second file is the joint definition of the architecture and *namd* workload.

You can execute the supplied python script ./a2wa.py to automatically convert all .cmp files in ./test directory to their WA-analogues.

# Hidden Parameters (require recompilation)

Several parameters are hard-coded, however they can be easily changed in the source code and the tool can be recompiled. This section summarizes the possible values of interest.

|  |  |  |
| --- | --- | --- |
| Parameter description | Defined in | Current definition |
| Maximum aspect ratio of the top-level mesh | src/arch/ParamIterator.cpp, GMeshDimIter::Advance() | double AR=2.0 |
| Size of the reply packets (bits) | src/arch/ArchPlanner.cpp, ArchPlanner::GenerateCurrentArchConfig() | int packetSize=512 |
| Delay of the interface between local IC and global mesh (cycles) | src/arch/ArchPlanner.cpp, ArchPlanner::GenerateCurrentArchConfig() | NiDelay=3 |
| Mesh router delay (cycles) | src/arch/ArchPlanner.cpp, ArchPlanner::GenerateCurrentArchConfig() | RouterDelay=3 |
| Ring router delay (cycles) | src/arch/ArchPlanner.cpp, ArchPlanner::GenerateCurrentArchConfig() | RouterDelay=1 |
| Ring link delay (cycles) | src/arch/ArchPlanner.cpp, ArchPlanner::GenerateCurrentArchConfig() | LinkDelay=0 |
| Latencies of memory controllers (cycles) | src/arch/ArchPlanner.cpp, ArchPlanner::GenerateCurrentArchConfig() | Latency=100 |

By default, *fixed-point iteration* and *bisection* methods are used to determine the approximate fixed point of the analytical dependency between latency and traffic. Using *subgradient* instead of bisection can improve the quality of estimation, at the cost of higher computational times. To choose between the methods, use the IterativePerfModel::*Run()* method in src/perf/IterativePerfModel.cpp.

# History

|  |  |  |
| --- | --- | --- |
| Version | Date | Comment |
| v0.2.1 | 04.Feb.2014 | - Added crossbar interconnects (only in clusters)  - Added constant functions  - Bug fix (output of IC type in SA/EO exploration) |
| v0.2 | 10.May.2013 | - Added support for multiple workloads  - Added calculation of ring router area when estimating total chip area  - Improved command line interface  - Added linkWidth parameter to generated .cmp files |
| v0.1 | 11.Dec.2012 | Initial release |

# Contacts

**Javier de San Pedro,** [**jspedro@lsi.upc.edu**](mailto:jspedro@lsi.upc.edu)

**Nikita Nikitin,** [**nikita.i.nikitin@gmail.com**](nikita.i.nikitin@gmail.com)