TPCalc : a throughput calculator for computer architecture studies

Pierre Michaud
IRISA/INRIA
pierre.michaud@inria.fr

Stijn Eyerman
Ghent University
Stijn.Eyerman@elis.UGent.be

Wouter Rogiest
Ghent University
wouter.rogiest@telin.ugent.be

July 9, 2014

Abstract

TPCalc is a throughput calculator for microarchitecture studies concerned with *multi-program workloads* consisting of *sequential* programs. Because microarchitecture simulators are slow, it is difficult to simulate throughput experiments (TPEX) where many jobs execute successively on a multicore. The usual practice of measuring instantaneous throughput on independent coschedules chosen more or less randomly is not a rigorous practice because it assumes that all the coschedules are equally important, which is not always true. TPCalc can compute the average throughput of a TPEX without actually simulating the TPEX. The user first defines the workload heterogeneity (number of different job types), the multicore configuration (number of cores and symmetries). TPCalc provides a list of base coschedules. The user then simulates these coschedules, using some benchmarks of his choice, and feeds back to TPCalc the measured execution rates (e.g., instructions per cycle or instructions per second).TPCalc eventually outputs the average throughput. Several throughput metrics are available, corresponding to different workload assumptions. The theory behind TPCalc can be found in our ACM TACO paper [1].

1 Throughput experiment (TPEX)

TPCalc allows to compute the **maximum average throughput** corresponding to a throughput experiment (TPEX). The sort of TPEX considered by TPCalc is the following:

- Jobs arrive at a rate greater than what the processor can execute (they wait in a queue).
- There are N different sorts of jobs in the TPEX. N is the workload heterogeneity degree, it is a parameter chosen by the user.
- The N job types are equally likely.
- The jobs are sequential programs.
- At any time, the number of jobs actually being executed is equal to the maximum number K of sequential threads that the processor can execute simultaneously.
- When a job is finished, the core on which it was running immediately starts executing a new job.

The average throughput of the TPEX is the total work executed divided by the total TPEX time. The total number of jobs executed is assumed to be very large, that is, TPCalc computes the asymptotic throughput. TPCalc can compute different throughput metrics corresponding to different workload assumptions.

Although TPCalc computes the average throughput of a TPEX, it does not actually simulate the TPEX. TPCalc considers all the possible *coschedules*, that is, all the possible combinations of K jobs. The average throughput is computed from the per-coschedule execution rates.

2 Using TPCalc

TPCalc is written in C++. It works on Unix systems (Linux, OS X, ...). To compile it with GCC:

```
g++ -O3 -std=c++0x -o tpcalc tpcalc.cpp
```

Or if you prefer Clang:

```
clang++ -O3 -stdlib=libc++ -o tpcalc tpcalc.cpp
```

TPCalc is used like a Unix command. It reads user data from the standard input and returns average throughput on the standard output. The general usage format is:

```
tpcalc JOBTYPES CONFIG [OPTIONS] [< DATAFILE]
```

- JOBTYPES is the number of different job types in the TPEX (i.e., the workload heterogeneity degree N). If JOBTYPES is set to 1, the workload is homogeneous. If JOBTYPES is greater than or equal to 2, the workload is heterogeneous.
- CONFIG is a string describing the multicore configuration, the number of cores and the symmetries. The multicore symmetry description language is explained in Section 5
- OPTIONS: the main options are
  - −**list** : tell TPCalc to output the list of base coschedules
  - −**metric** M : tell TPCalc to compute throughput metric M (see Table 1)
  - −**ref** R_0 R_1 ... R_{N-1} : specify the N=JOBTYPES reference execution rates (default R_i = 1)
- DATAFILE is a file containing the execution rates (typically, IPC or IPS values) obtained from simulations.

3 Example

Suppose we want to evaluate the maximum throughput of a 4-thread SMT core running workloads consisting of 2 different sorts of jobs. We type the following command:

```
tpcalc 2 "4"
```

for a heterogeneity degree of 2 and 4 interchangeable logical cores. The output of this command is

```
4 logical cores
5 base coschedules
These cores are identical: 0 1 2 3
```

metric's name	unit of work	type of workload	TPEX assumptions
AVI	instruction	variable	all coschedules equally likely in time
AVW	weighted instruction	variable	all coschedules equally likely in time
AFI	instruction	fixed	jobs execute equal number of instructions all job types equally likely
AFW	weighted instruction	fixed	jobs execute equal number of weighted instructions all job types equally likely

Table 1: **List of average throughput metrics that TPCalc can compute (see reference [1]). All JOB-TYPES job types are equally likely. We do not recommend using the AVI and AVW metrics unless a variable workload is really what you want.**

On this example, there are 5 base coschedules. We can obtain the list of base coschedules with the following command:

```
tpcalc 2 "4" -list
```

which outputs

```
0 :  0  0  0  0
1 :  1  0  0  0
2 :  1  1  0  0
3 :  1  1  1  0
4 :  1  1  1  1
```

Here, "0" represents one job type and "1" represents the other job type. Each of the 4 columns corresponds to one particular logical core. For instance, coschedule #3 executes a job of type "1" on logical cores #0, #1 and #2, and a job of type "0" on logical core #3. Notice that, in this example, all the cores are interchangeable: however we permute the jobs, the job execution rates remain unchanged. Hence we only need to simulate 5 base coschedules instead of all $2^4 = 16$ coschedules.

We just need to replace the "0" and "1" with some benchmarks of our choice (or some execution phases extracted from benchmarks). For example, let's say that we have 4 benchmarks named A, B, C and D. We can form 6 different 2-heterogeneous workloads from these 4 benchmarks: AB, AC, AD, BC, BD and CD. Let's say we are interested by workload AB. We simulate the 5 base coschedules formed from workload AB, and we measure the IPCs, i.e., the number of instructions executed per cycle by each of the 4 jobs in each coschedule. Table 2 gives some example IPC values. Here we have 5 base coschedules and 4 logical cores, hence a total of 20 IPC values, that we feed to TPCalc through the standard input. For example, to compute the AFI throughput:

```
tpcalc 2 "4" -metric AFI
1 1 1 1
.5 .9 .9 .9
.5 .5 .9 .9
.4 .4 .4 .8
.4 .4 .4 .4
```

	core 0	core 1	core 2	core 3
coschedule 0	**A**	**A**	**A**	**A**
IPC	1	1	1	1
coschedule 1	**B**	**A**	**A**	**A**
IPC	0.5	0.9	0.9	0.9
coschedule 2	**B**	**B**	**A**	**A**
IPC	0.5	0.5	0.9	0.9
coschedule 3	**B**	**B**	**B**	**A**
IPC	0.4	0.4	0.4	0.8
coschedule 4	**B**	**B**	**B**	**B**
IPC	0.4	0.4	0.4	0.4

Table 2: Example IPC values for a 2-heterogeneous workload consisting of job types A and B (Section 3)

which gives 2.2828 instructions per cycle. Notice that the IPC values are entered in one-to-one correspondence with the list of coschedules given by the **−list** option. Equivalently, we can feed the IPC values to TPCalc through a Unix pipe:

```
echo 1 1 1 1 .5 .9 .9 .9 .5 .5 .9 .9 .4 .4 .4 .8 .4 .4 .4 .4 | tpcalc
2 "4" -metric AFI
```

But the more convenient solution is probably to write the IPC values in a file, say IPCFILE, and redirect the standard input:

```
tpcalc 2 "4" -metric AFI < IPCFILE
```

The AFI throughput is expressed in the same unit as that of the input execution rates. In this example, the execution rates are IPCs (instructions per cycle), hence the AFI is also in instructions per cycle (if the execution rates were in billion instructions per second, the AFI throughput would be in billion instructions per second).

In computer architecture studies, we typically compute the throughput for several workloads. Four benchmarks, as in this example, lead to 6 different 2-heterogeneous workloads, each workload corresponding to a distinct TPEX (for a given metric). When there is a large number of possible workloads, evaluating the throughput of all of them may require too much simulation time. In this case, the usual solution is to sample the workload space [2, 3]. Then, one can summarize the workload space by computing a measure of central tendency, such as the arithmetic mean of per-workload throughputs, or the median throughput.

4 Weighted-instruction throughput

The instruction throughput metrics, AVI and AFI, assume that one instruction of any benchmark is worth one instruction of any other benchmark in terms of average quantity of work executed. Moreover, the AFI metric assumes that all the jobs execute the same quantity of work, hence the same number of instructions.

The weighted-instruction throughput metrics, AVW and AFW, weight the instructions of different benchmarks differently: the instructions of a benchmark have a weight inversely proportional to the reference execution rate of that benchmark. The reference execution rate is the execution rate of the benchmark when running alone on a reference machine. For instance, if benchmark A has a reference IPC of 1 and benchmark B a reference IPC of 2, one instruction of benchmark A is worth 2 instructions of benchmark B in terms of

average quantity of work executed. Moreover, the AFW metric assumes that all the jobs execute the same quantity of work, which means that all the jobs have the same execution time when running alone on the reference machine.

In the example of Section 3, a natural choice for defining reference IPCs would be to run benchmarks A and B in isolation on the processor being considered, that is, we choose as reference machine the machine whose throughput we seek to evaluate. In this situation, the reference IPC for a benchmark is typically greater than or equal to its IPCs in the coschedules, because running the benchmark in isolation allows it to use all the processor resources. For instance, let's assume that the reference IPC of benchmark A is 1 and that of B is 2. We can compute the AFW throughput as follows:

```
tpcalc 2 "4" -ref 1 2 -metric AFW
1 1 1 1
.5 .9 .9 .9
.5 .5 .9 .9
.4 .4 .4 .8
.4 .4 .4 .4
```

which gives 1.3190 weighted instructions per cycle. In fact, there is no fundamental difference between instruction throughput and weighted instruction throughput. Computing AFW with the same reference execution rate for all job types is equivalent to AFI, and computing AFI by replacing IPCs with speedups is equivalent to AFW. For instance,

```
tpcalc 2 "4" -metric AFI
1 1 1 1
.25 .9 .9 .9
.25 .25 .9 .9
.2 .2 .2 .8
.2 .2 .2 .2
```

gives 1.3190. Nevertheless, the $-$**ref** option is convenient for computing instruction throughput and weighted-instruction throughput simultaneously, for instance:

```
tpcalc 2 "4" -ref 1 2 -metric AFI -metric AFW
1 1 1 1
.5 .9 .9 .9
.5 .5 .9 .9
.4 .4 .4 .8
.4 .4 .4 .4
```

which gives

```
2.2828 1.3190
```

Reference execution rates. The reference execution rates are part of the definition of the AVW/AFW metrics. Different reference execution rates lead to different AVW/AFW metrics. The reference execution rates are typically defined from isolated execution on a reference machine. But other methods are possible. For instance, reference execution rates could be defined from the execution of homogeneous coschedules.

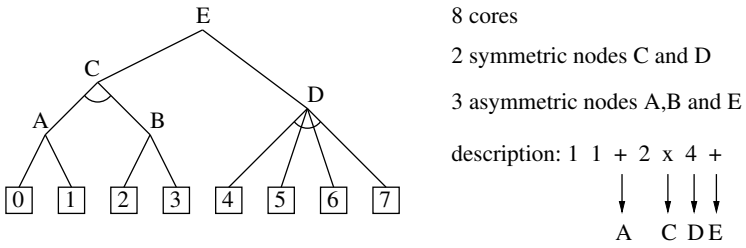

Figure 1: Tree-based multicore symmetry description language. Example of heterogeneous 8-core processor. Nodes A and B are identical, cores 4,5,6,7 are identical. The description language uses reverse polish notation.

5 Multicore symmetry description language

So far we have considered multicore examples where logical cores are interchangeable. Taking advantage of multicore symmetries allows to reduce considerably the number of base coschedules. However, in practice, multicores are generally not fully symmetric. Still, it is important to exploit the existing symmetries for having a reasonable number of base coschedules.

The multicore symmetry description language used by TPCalc is based on a tree representation. Figure 1 shows an example of heterogeneous 8-core processor. The logical cores are the leaves of the tree. They are numbered from left to right, from 0 to 7. This ordering of logical cores, from left to right, is the one assumed by the $-$**list** and $-$**metric** options. The tree nodes correspond to multicore nodes, i.e., groups of cores. Logical cores #4, #5, #6 and #7 are grouped in node D, which is a symmetric node, meaning that the 4 cores are interchangeable (for instance, node D could be a 4-thread SMT physical core). Node A is an asymmetric node containing 2 different cores, core #0 and core #1. Node C is a symmetric node formed of nodes A and B, meaning that nodes A and B are identical and interchangeable: if at the same time we permute the jobs on cores #0 and #2 and we permute the jobs on cores #1 and #3, the 4 jobs have their execution rates unchanged. Node E is the grouping of nodes C and D, it represents the whole multicore. Node E is necessarily asymmetric here because nodes C and D have a different structure.

The multicore symmetry description language uses Reverse Polish Notation to describe the tree. The string describing the tree shown in Figure 1 is

```
"1 1 + 2 x 4 +"
```

Asymmetric nodes are created with the symbol "+", which groups 2 previously created nodes (or cores) in a new node. Simple symmetric nodes like D are created by specifying an integer value greater than one, representing the number of cores in the node. Complex symmetric nodes like C are created with the symbol "x", which creates several identical instances of a previously created node and groups them in a new node. TPCalc evalutes the description string sequentially, using a stack. In the example of Figure 1, **"1 1 +"** creates node A, **"2 x"** duplicates node A and creates node C, **"4"** creates node D and the final **"+"** creates node E. For obtaining the number of coschedules corresponding to a 2-heterogeneous workload, we use the following command

```
tpcalc 2 "1 1 + 2 x 4 +"
```

whose output is

hardware configuration	number of logical cores	description string
2-thread SMT core	2	`"2"`
2 different cores	2	`"1 1 +"`
3 different cores	3	`"1 1 + 1 +"`
4 different cores	4	`"1 1 + 1 + 1 +`
1 core of type X and 4 interchangeable cores of type Y	5	`"1 4 +"`
3 interchangeable 2-thread SMT cores	6	`"2 3 x"`
3 interchangeable nodes, 2 different cores per node	6	`"1 1 + 3 x"`
2 interchangeable nodes, 3 interchangeable 4-thread SMT cores per node	24	`"4 3 x 2 x"`
3 interchangeable 2-thread SMT cores and 2 interchangeable 4-thread SMT cores	14	`"2 3 x 4 2 x +"`

Table 3: Examples of multicore configuration and their description in TPCalc tree-based language.

```
8 logical cores
50 base coschedules
These cores are identical: 0 2
These cores are identical: 1 3
These cores are identical: 4 5 6 7
```

Table 3 provides some examples of multicore configurations and their description in the tree-based language. It should be noted that TPCalc tree-based language does not allow to express all possible kinds of symmetries. But it should be sufficient for most studies.

6 Limitations and recommendations

When possible, the AFI and AFW metrics should be used in computer architecture studies concerned with multi-program workloads consisting of sequential programs. We do not recommend using the AVI and AVW metrics unless a variable workload is really what you want.

6.1 Monte Carlo method

The TPCalc software computes the AFI and AFW metrics by solving a Markov chain. To validate the software, we also implemented a Monte Carlo method for computing the AFI and AFW metrics. The Monte Carlo method provides an approximation of AFI and AFW. In case users have doubts about an AFI value, they can invoke the Monte Carlo method with the `-metric AFI~` option (AFI with a tilde). For example,

```
tpcalc 2 "4" -metric AFI -metric AFI~ -mcsimexp -mcsim 1000000
1 1 1 1
.5 .9 .9 .9
.5 .5 .9 .9
```

```
.4 .4 .4 .8
.4 .4 .4 .4
```

gives

```
2.2828 2.2859
```

The `-mcsimexp` option tells TPCalc to use exponentially distributed job sizes (by default, the Monte Carlo method assumes fixed size jobs). The `-mcsim 1000000` option tells TPCalc to simulate the execution of 1 million jobs (by default, the Monte Carlo method assumes 10000 jobs). Similarly, the Monte Carlo method for the AFW metric can be invoked with the `-metric AFW~` option.

6.2 Large number of base coschedules

Because the number of base coschedules increases quickly with the number of logical cores when the multicore is not fully symmetric, the number of base coschedules may become so large that a rigorous evaluation of the AFI/AFW throughput becomes infeasible. In such a case, TPCalc can still be useful, provided some approximations are made.

For example, the IBM POWER7 processor features 8 interchangeable cores, each core being 4-thread SMT, for a total of 32 logical cores [4]. However, core resource sharing between the 4 threads is complex[1] and the 4 logical cores are not completely interchangeable. Therefore, the description string for the POWER7 multicore is

```
"1 1 + 2 x 8 x"
```

which leads to 24310 base coschedules assuming a heterogeneity degree of just 2.

Nevertheless, evaluating multi-program throughput is generally not a goal by itself. In computer architecture studies, it is a means for studying resource sharing (is it better to share a resource or replicate it ? what is the best way to manage a shared resource ?). In practice, it is often possible to simplify the problem.

If we want to study core resource sharing, we probably do not need to consider the 8 cores simultaneously to identify good design decisions. We can evaluate the AFI/AFW throughput of a single core:

```
"1 1 + 2 x"
```

If we want to study last-level cache (LLC) sharing, we can conjecture that an LLC implementation is good (or bad) independently of how core resource sharing is precisely implemented, and assume fully symmetric cores:

```
"4 8 x"
```

which leads to ”only” 495 base coschedules assuming a heterogeneity degree of 2. Our ACM TACO paper discusses some alternatives to the AFI and AFW metrics in case the number of base coschedules is too large [1].

[1] When 4 threads are running on the POWER7 core, the issue queue, integer pipelines and integer physical registers are split in two partitions: integer instructions of threads 0 and 1 go in one partition, integer instructions of threads 2 and 3 go in the other partition [4]. The floating-point and vector physical registers are split in two partitions, one partition for threads 0 and 2, the other partition for threads 1 and 3.

References

[1] S. Eyerman, P. Michaud, W. Rogiest. Multi-program throughput metrics: a systematic approach. *ACM Transactions on Architecture and Code Optimization* (TACO), 2014.

[2] K. Van Craeynest, L. Eeckhout. The multi-program performance model: debunking current practice in multi-core simulation. *IEEE International Symposium on Workload Characterization* (IISWC), 2011.

[3] R. A. Velásquez, P. Michaud, A. Seznec. Selecting benchmark combinations for the evaluation of multicore throughput. *IEEE International Symposium on Performance Analysis of Systems and Software* (ISPASS), 2013.

[4] B. Sinharoy et al. IBM POWER7 multicore server processor. *IBM Journal of Research & Development*, vol. 55, No. 3, May/June 2011.