CPU+GPU Load Balance Guided by Execution Time Prediction

Jean-François Dollinger, Vincent Loechner

Inria CAMUS, ICube Lab., University of Strasbourg

jean-francois.dollinger@inria.fr, vincent.loechner@inria.fr

19 January 2015

Outline

Achieving and predicting performance on CPU/GPU is difficult.
Sensitive to:

- Input dataset (CUDA grid size, cache effects)
- Compiler optimizations (unrolling, fission)
- Cloudy infrastructures
- Hardware availability
- Efficient resources exploitation

Because of dynamic behaviors compilers miss performance opportunities

- PLUTO
- PPCG
- Par4All
- openACC/HMPP: manual tuning

$\rightarrow$ Automatic methods are the way to go (typical use case)
$\rightarrow$ Our interest: polyhedral codes

How to get performance?

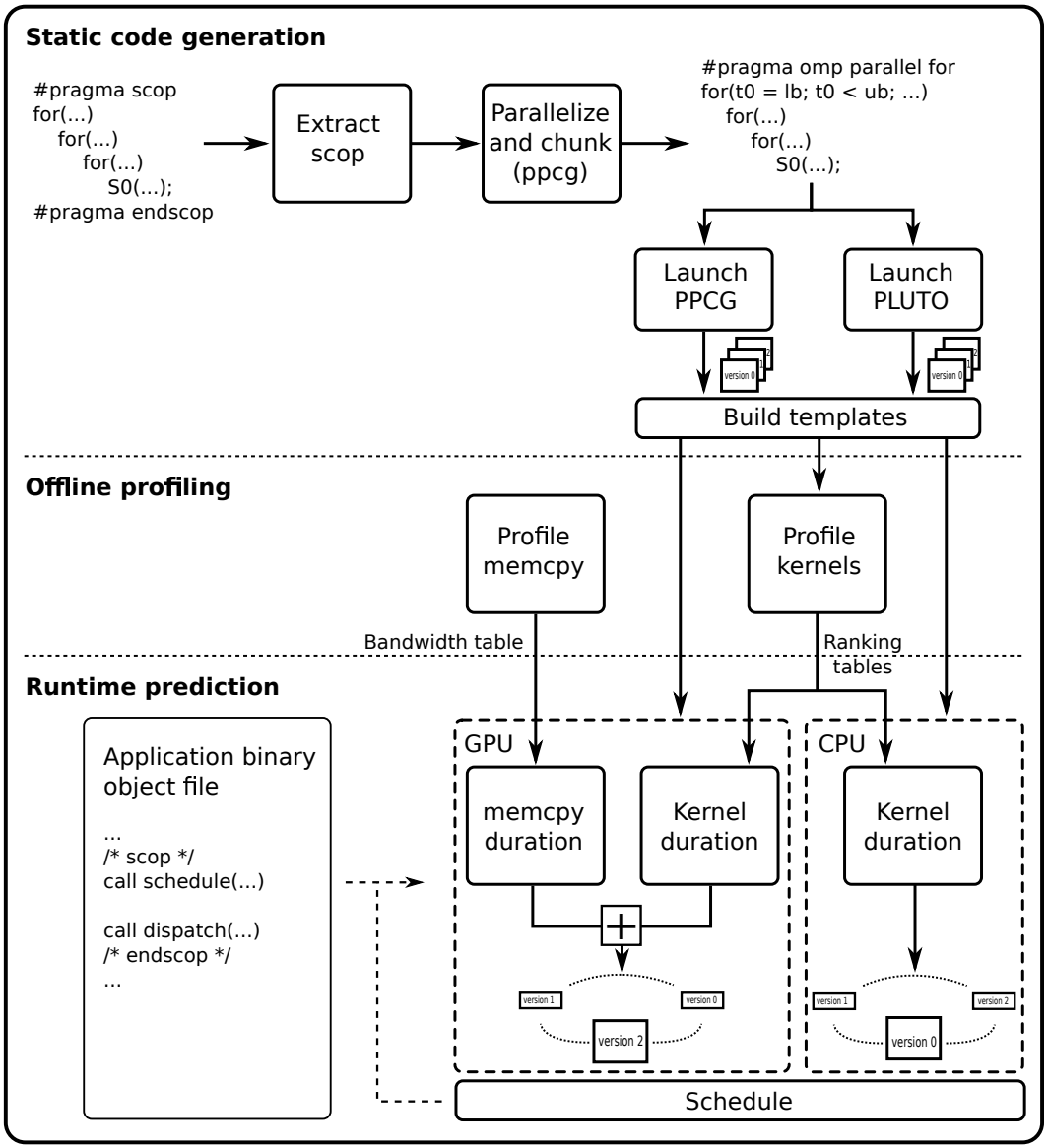
- Right code with right PU (Processing Unit)
- Select best code version on each given PU
- Ensure load balance between PUs

$\rightarrow$ Multi-versioning + runtime code selection = win

Outline

Multi-versioning: performance factors

- Static factors (instruction)
- External dynamic factors (scheduler)
- Internal dynamic factors (cache effects, memory contention)


Static code generation
#pragma scop
for(...)
for(...)
for(...)
S0(...);
#pragma endscop
Extract scop
Parallelize and chunk (ppcg)
#pragma omp parallel for
for(t0 = lb; t0 < ub; ...)
for(...)
for(...)
S0(...);
Launch PPCG
Launch PLUTO
version 0
version 0
Build templates
Offline profiling
Profile memcpy
Profile kernels
Bandwidth table
Ranking tables
Runtime prediction
Application binary object file
...
/* scop */
call schedule(...)
call dispatch(...)
/* endscop */
...
GPU
memcpy duration
Kernel duration
version 1
version 0
version 2
CPU
Kernel duration
version 1
version 2
version 0
Schedule

Pedro Framework [Benoit Pradelle et al. 2011]

- Multi-versioning of polyhedral loop nests
- Target : multicore CPUs

Outline

Code version

- Block size
- Tile size
- Schedule

$\rightarrow$ controlled by PPCG cmd line options

PPCG, source-to-source compiler

- Transforms C to CUDA
- Generates:
  - Ehrhart polynomials
  - Loop nest parameters

Python scripts

- Fill templates in C code

Outline

Data transfers: host $\leftrightarrow$ device

- Parameter: message size
- Asymetric and non-uniform bandwidth

Code simulation

- Parameters: number of CUDA blocks, sequential parameters
- Load balance
- Memory contention

How to model the performance curves ?

- Affine intervals detection

1st test platform

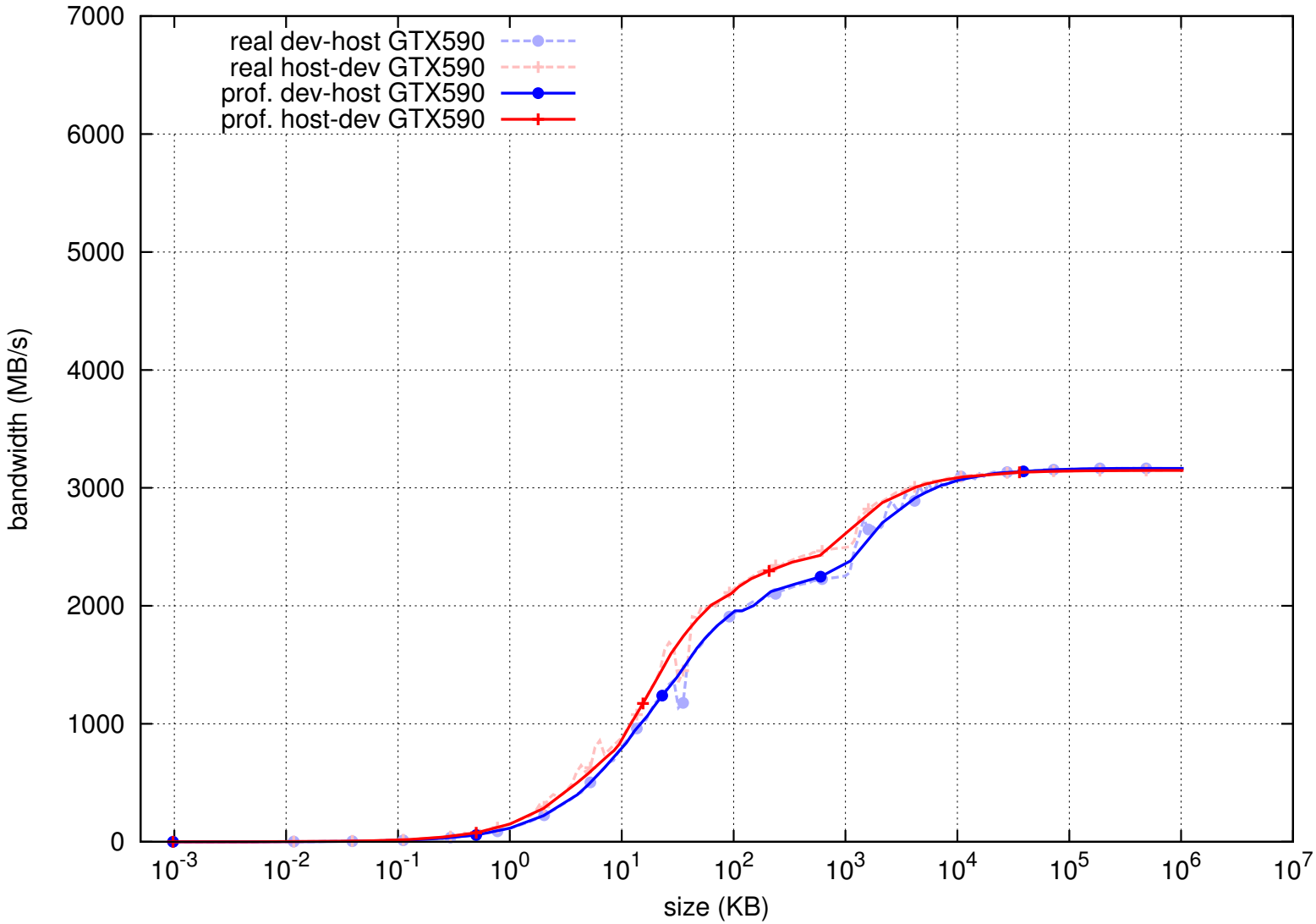
- 2 Nvidia GTX 590 (16 (SM) * 32 (SP))
- Asus P8P67-Pro (PCIe 2, x8 per card)
- Core i7 2700k, stock

2nd test platform

- Nvidia GTX 680 (8 (SM) * 192 (SP))
- Asus P8P67-Deluxe (PCIe 2, x16)
- Core i7 2600

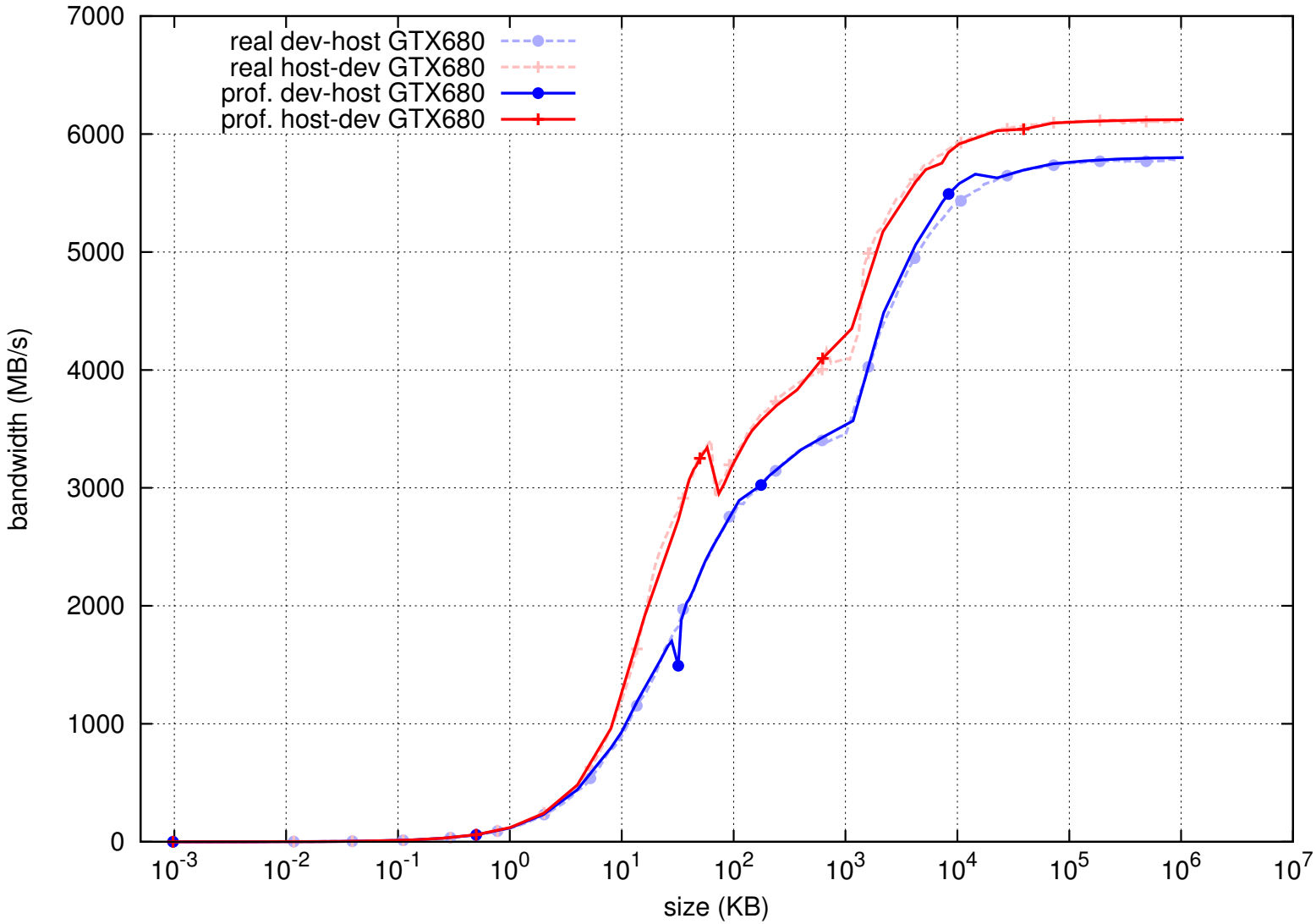
Prediction

Data transfers (testbed 1)



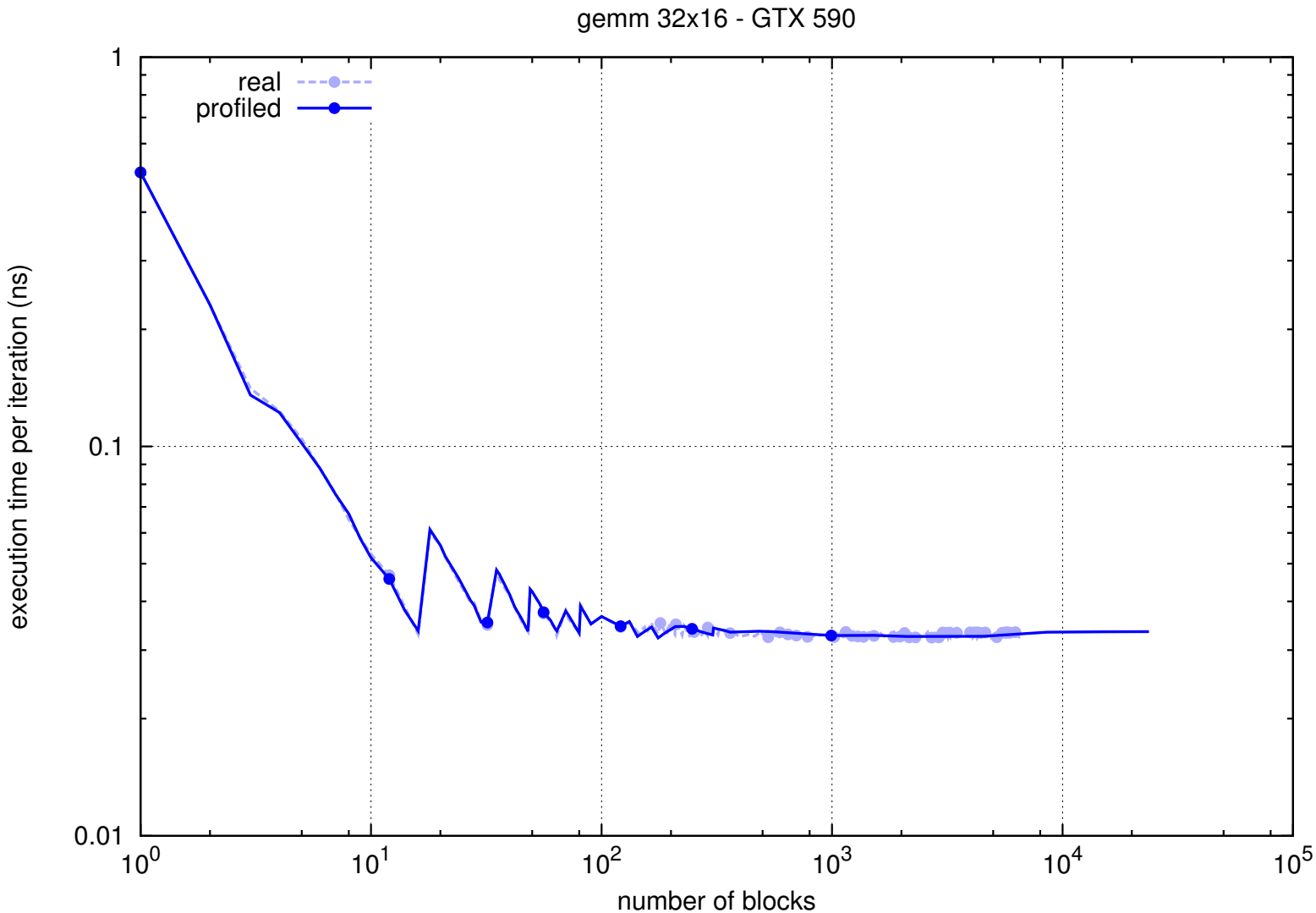
Prediction

Data transfers (testbed 2)



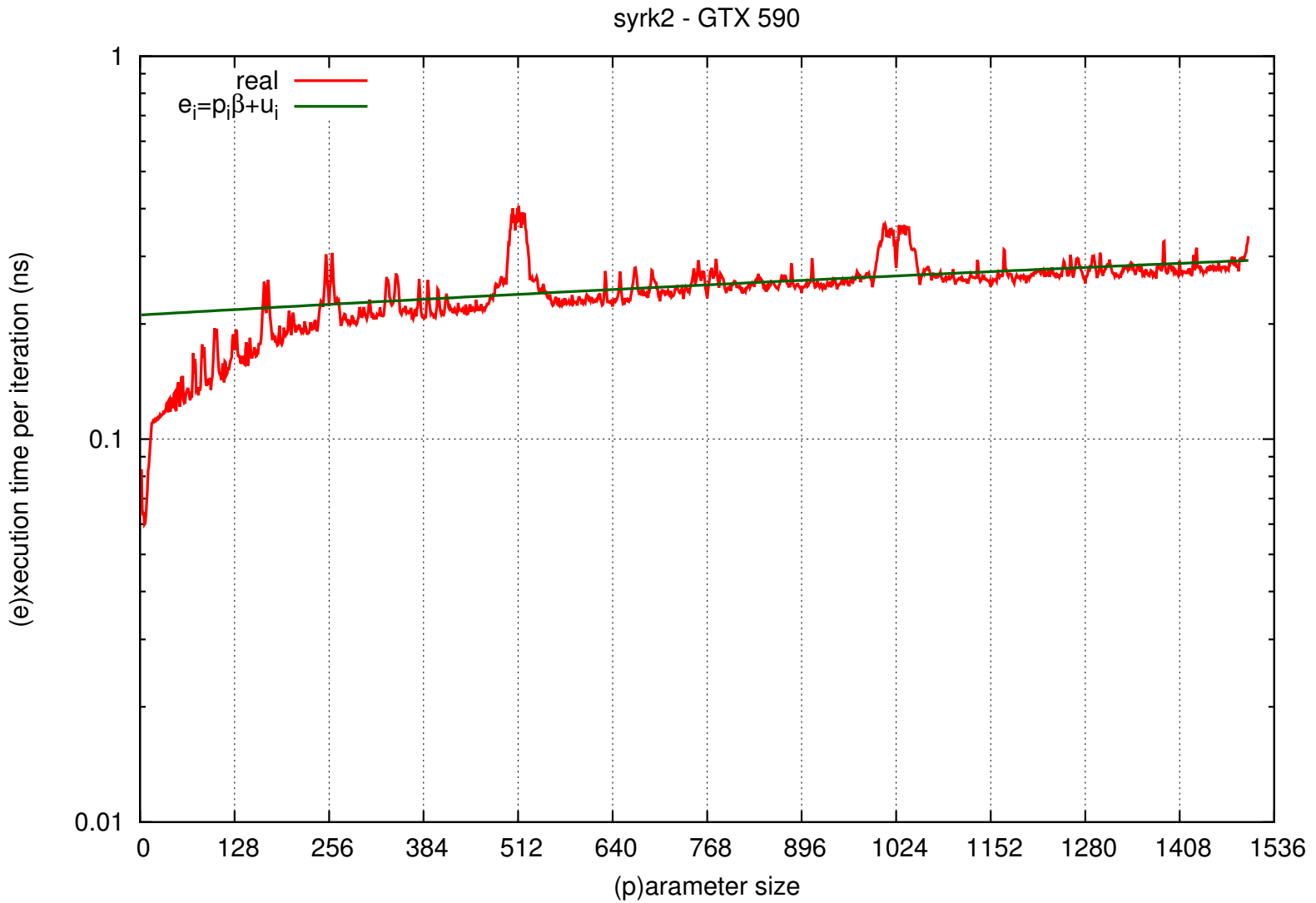
Prediction

Kernel simulation (testbed 1)



Prediction

Kernel simulation (testbed 1)



Outline

Outermost parallel loop split into chunks

- Each chunk associated to one PU
- PUs performance differ

$\rightarrow$ Ensure load balance

Multi-Versioning

- Code optimized towards target (PLUTO + PPCG)
- Multiple code versions (combined)

Two components:

- Scheduler:
  - Execution time of chunks [B. Pradelle et al.] + [J-F. Dollinger et al.]
  - Adjust chunks sizes
- Dispatcher

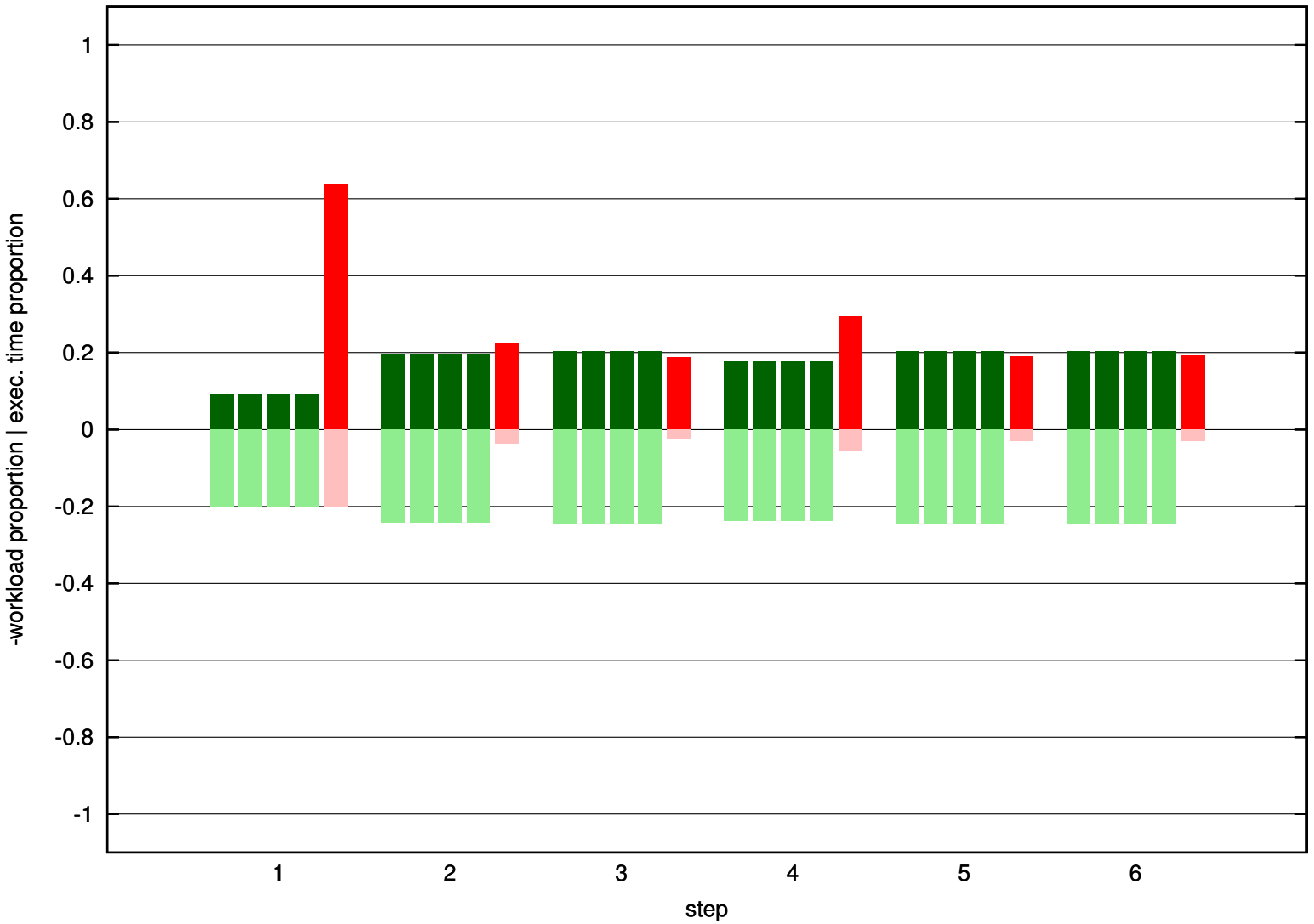
Scheduler functioning

1. $T_0 = t_0 * Card\ D_0 \approx t_1 * Card\ D_1 \approx ... \approx t_n * Card\ D_n$
2. T_i must tend to $1/n * \sum_{i=0}^{n-1}(t_i * Card\ D_i) = 1/n * T_{all}$
3. $t_i = f(G_i, \overline{seq})$ on GPU
4. $t_i = g(P_i, S_i)$ on CPU

Eliminate inefficient PUs

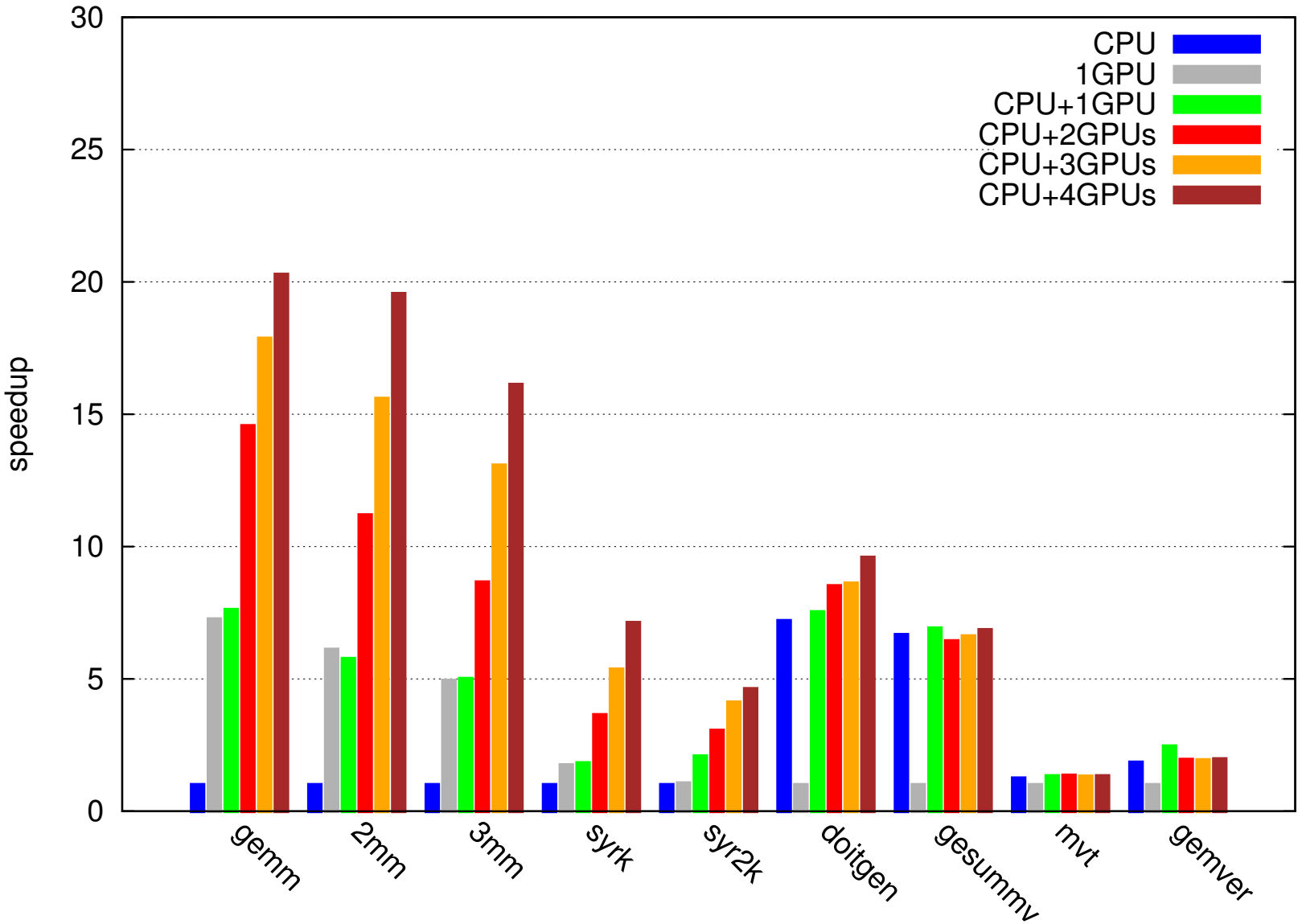
Runtime

CPU + GPU



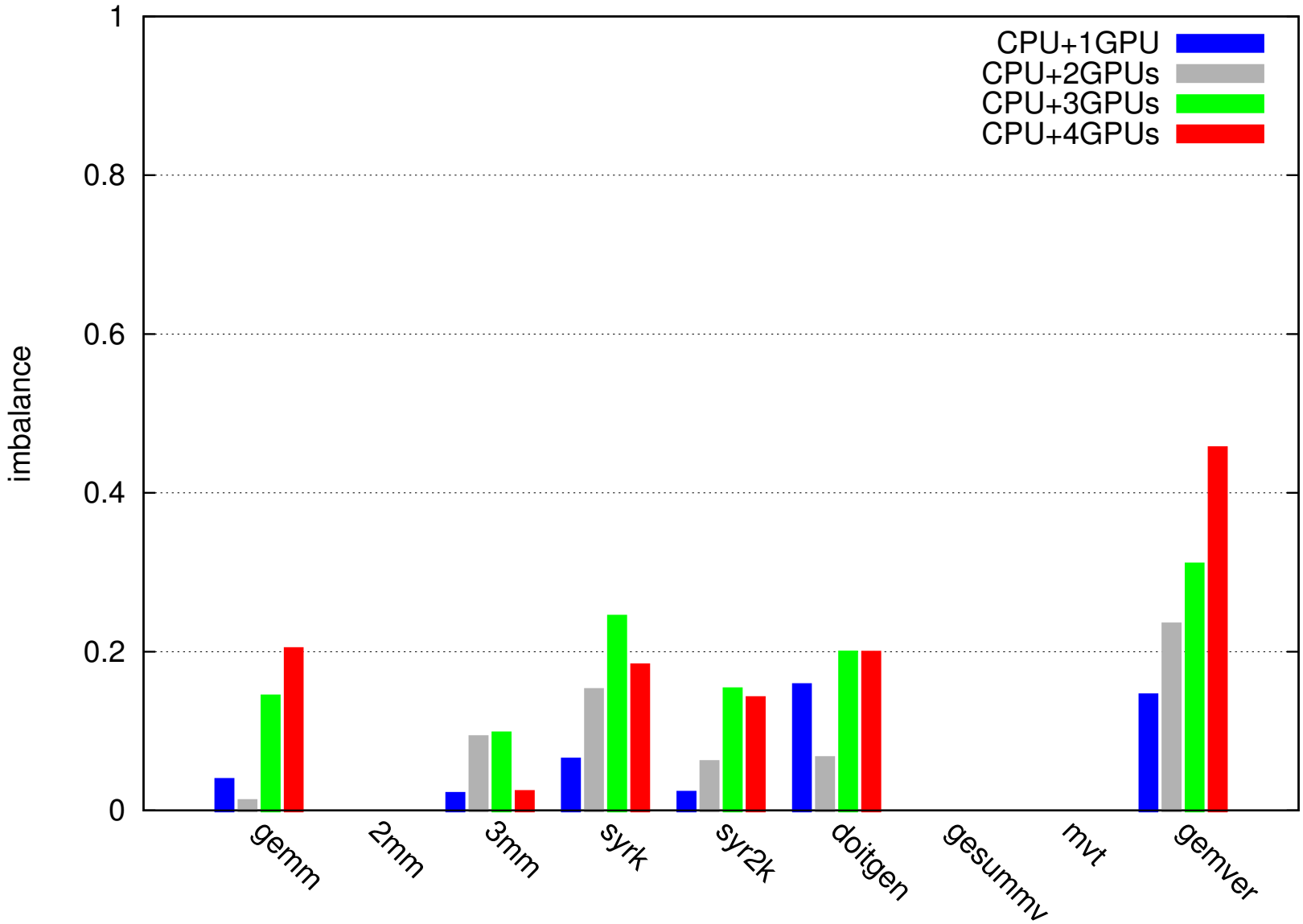
Runtime

CPU + GPU (speedup to one PU)



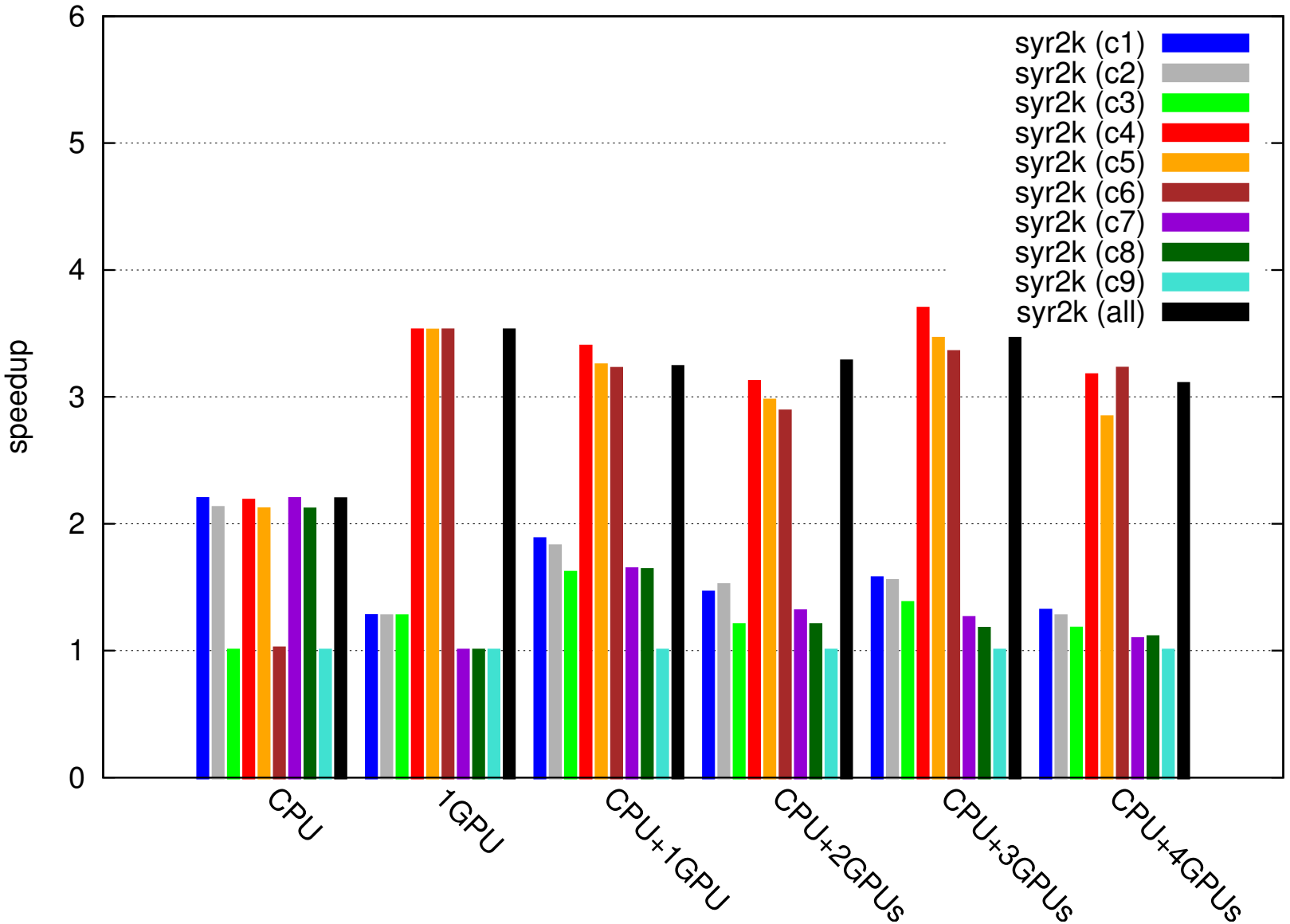
Runtime

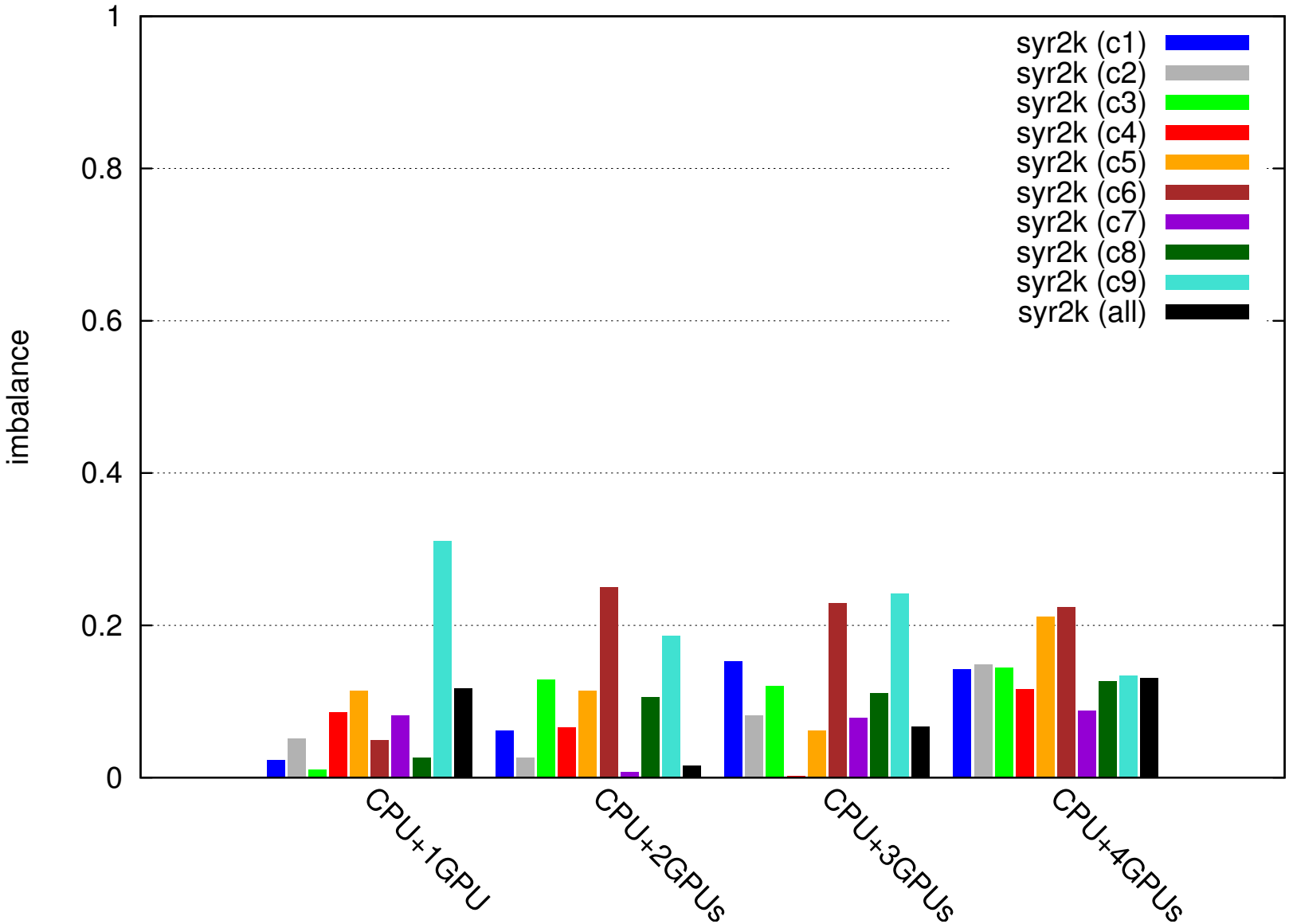
CPU + GPU (load imbalance)



Runtime

Multiversioning CPU + GPU (speedup to worst)




imbalance
1
0.8
0.6
0.4
0.2
0
syr2k (c1)
syr2k (c2)
syr2k (c3)
syr2k (c4)
syr2k (c5)
syr2k (c6)
syr2k (c7)
syr2k (c8)
syr2k (c9)
syr2k (all)
CPU+1GPU
CPU+2GPUs
CPU+3GPUs
CPU+4GPUs

Framework capabilities

- Execution time prediction
- Fastest version selection
- CPU vs GPU competition
- CPU + GPU joint usage

Future work

- Energy consumption

Outline

Offline profiling: ranking table

Number of threads	version 1	version 2	version 3
1	40 ms	55 ms	32 ms
2	32 ms	28 ms	17 ms
3	22 ms	15 ms	9 ms
4	14 ms	7 ms	8 ms

Online prediction: execution time computation

$$
\begin{aligned}
\mathit{observation} &= \{2000, 600, 300, 300\} \\
\mathit{prediction}\ (\mathit{version}1) &= ((2000 - 600) * 40) + ((600 - 300) * 32) \\
&\quad + (0 * 22) + (300 * 14) \\
&= 69800\mathit{ms}
\end{aligned}
$$

The algorithm stages:

- Init.: distribute iterations equitably amongst PUs
- Repeat 10 times:
  - Compute per chunk execution time
  - $r_i = T_i / T_{all}$
  - Adjust chunk size according to r_i