A Proposal for Autotuning Linear Algebra Routines on Multicore Platforms

Javier Cuenca, Luis P. García
Domingo Giménez

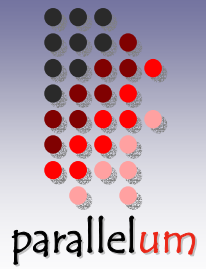
Parallel Computing Group
University of Murcia, SPAIN

parallelum

Introduction

- The arrival of multicores: parallel computing more available to the scientific groups
- To describe parallelism to the compiler: OpenMP, a directive-based syntax
- Problem:
  - there are no standardized OpenMP compilers
  - number of threads → the performance greatly varies
- Our goal: a Poly-Compilation Engine (PCE)
  - generates different executables of each routine (one for each compiler in the system)
  - selects the executable which best fits the problem characteristics

Outline



- Introduction
- Our approach: Poly-Compilation Engine (PCE)
  - PCB: Benchmarking Routines of the PCE
  - Proof of concept of the PCE
- Conclusions

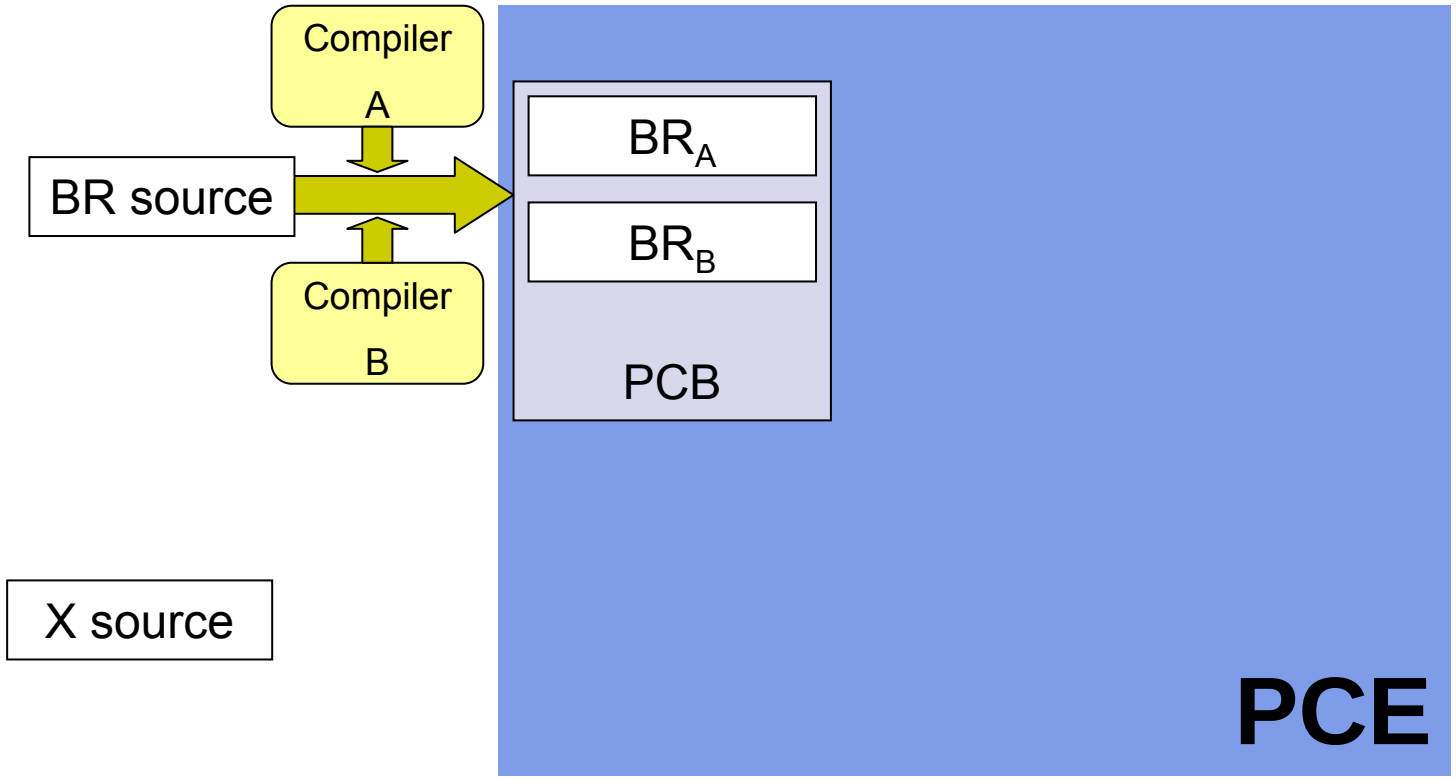
Our approach:
Poly-Compilation Engine (PCE)

BR source

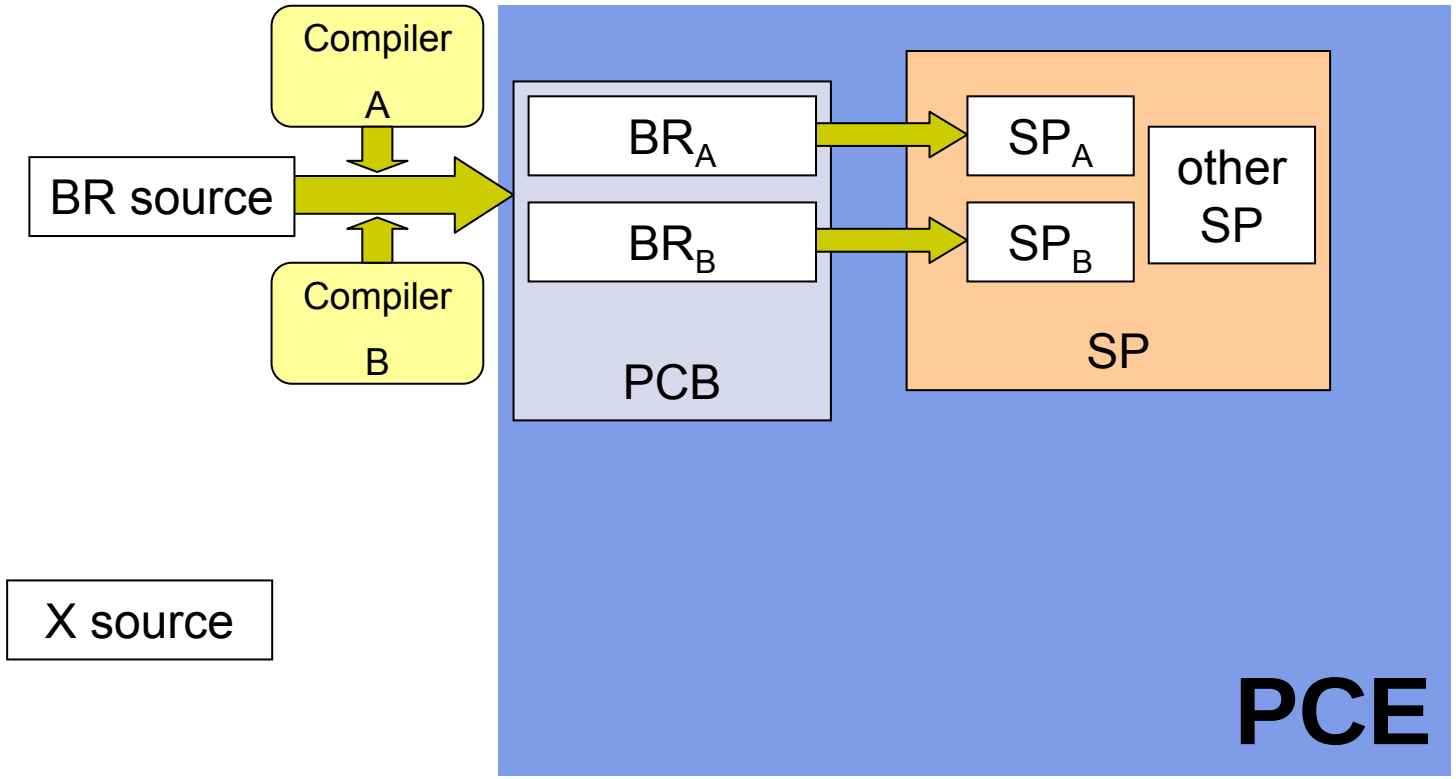
X source

PCE

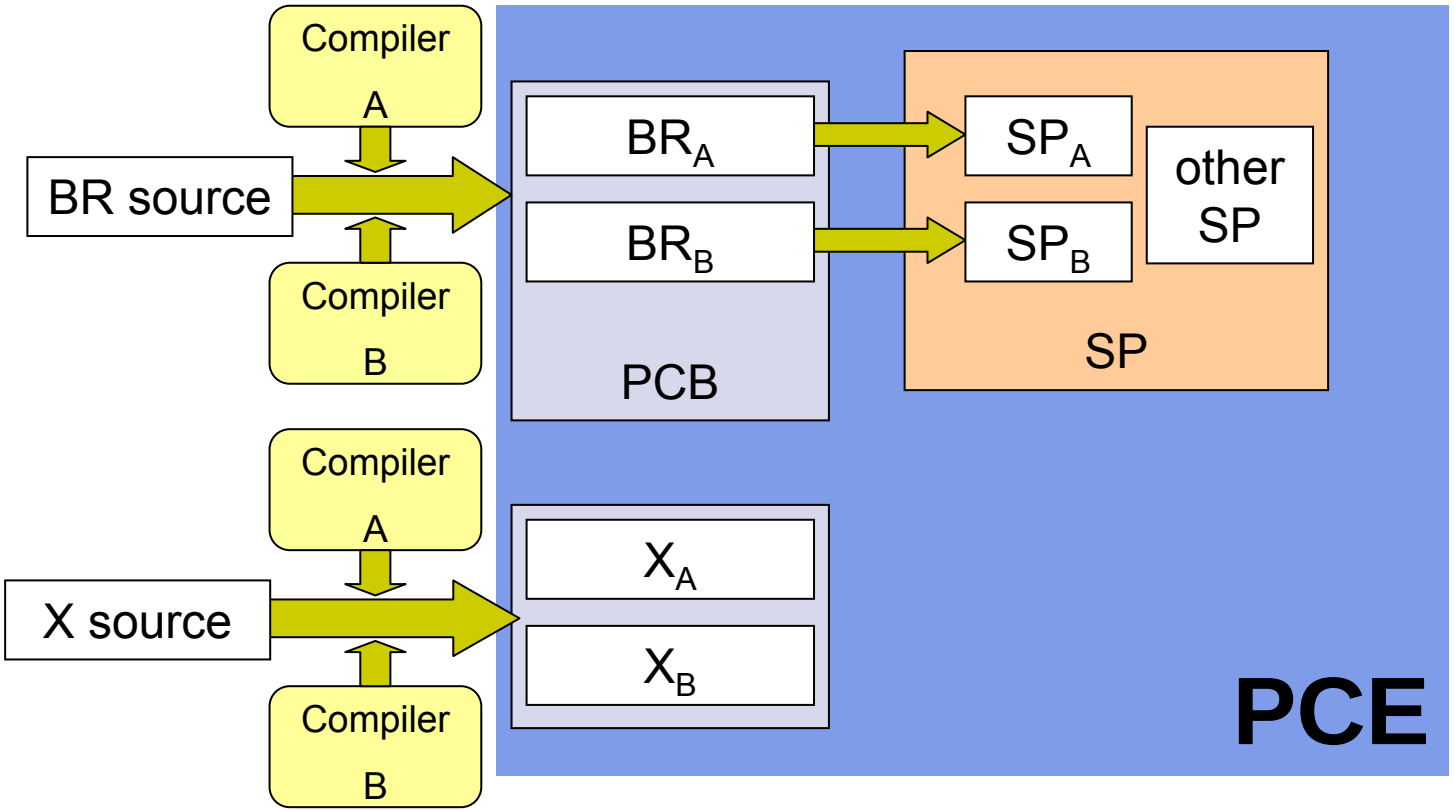
Our approach:
Poly-Compilation Engine (PCE)



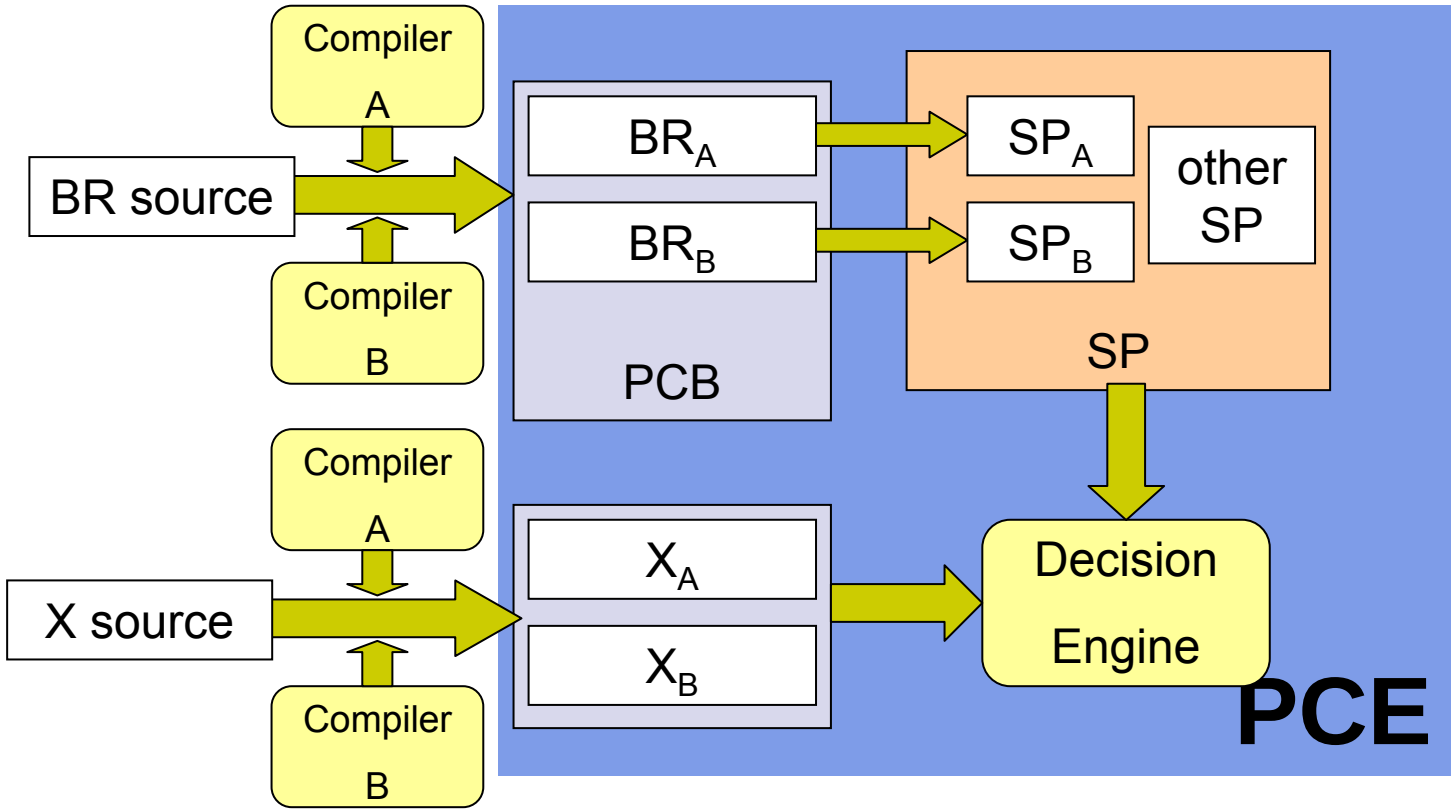
Our approach:
Poly-Compilation Engine (PCE)



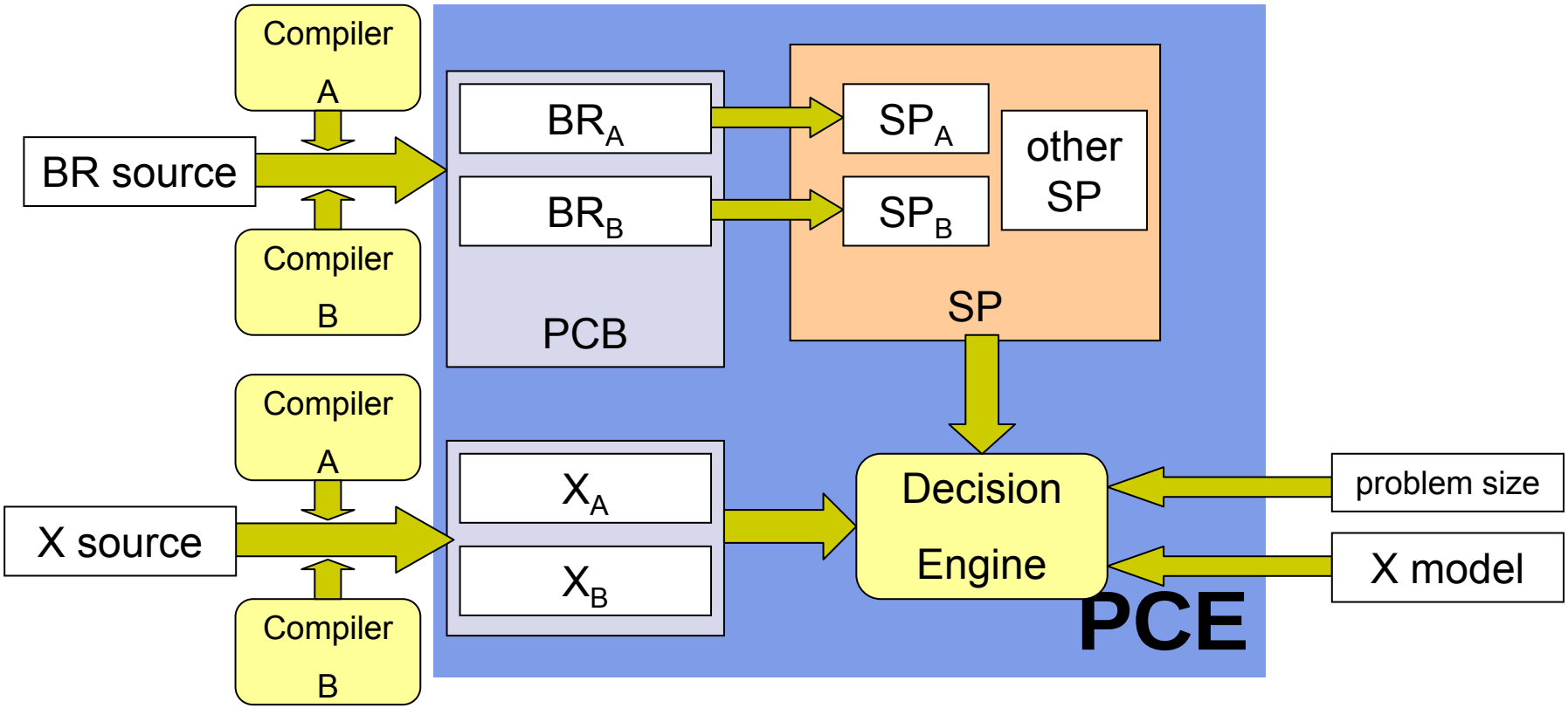
Our approach:
Poly-Compilation Engine (PCE)



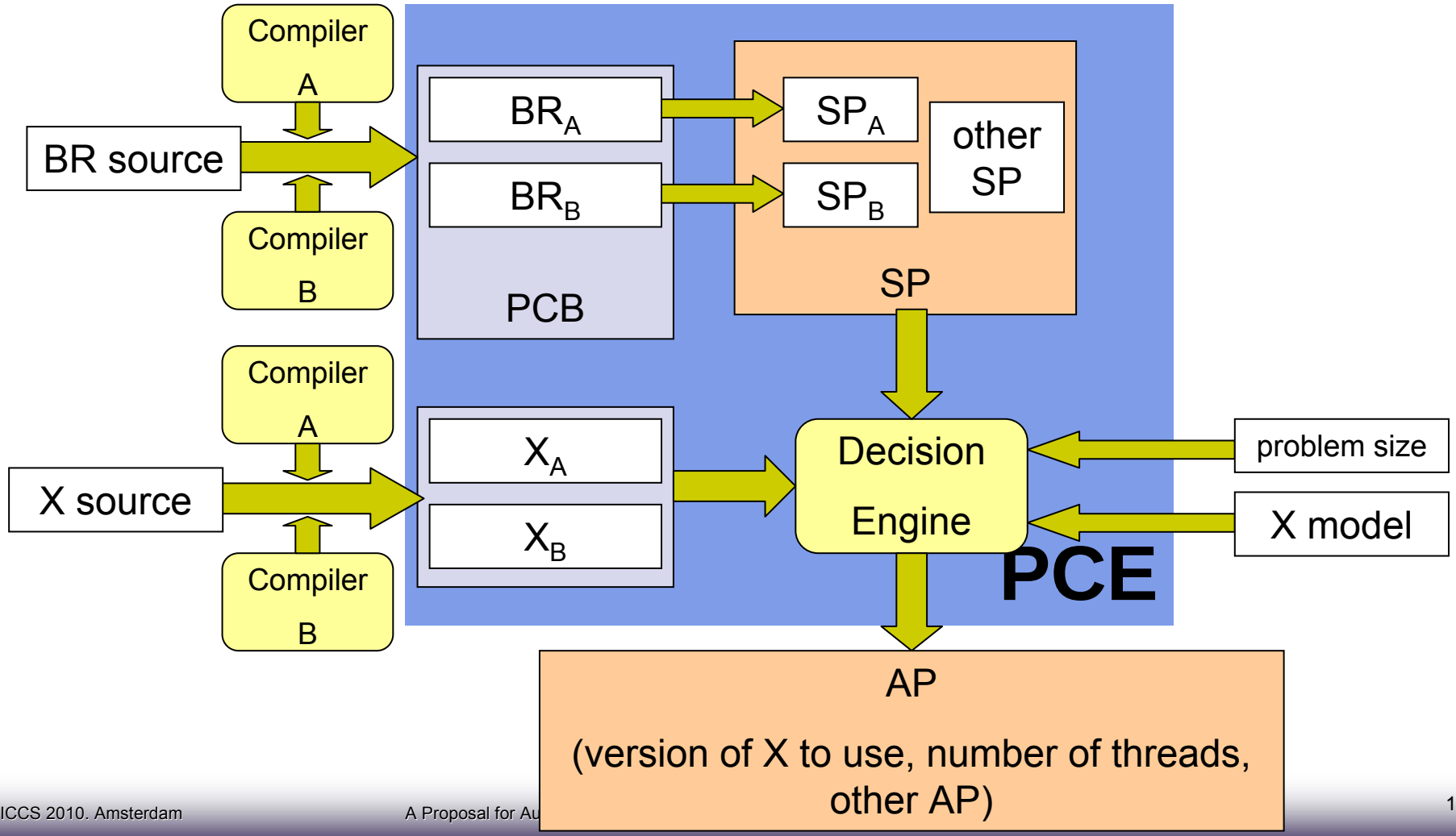
Our approach:
Poly-Compilation Engine (PCE)



Our approach:
Poly-Compilation Engine (PCE)



Our approach:
Poly-Compilation Engine (PCE)



Outline

- Introduction
- Our approach: Poly-Compilation Engine (PCE)
  - PCB: Benchmarking Routines of the PCE
  - Proof of concept of the PCE
- Conclusions

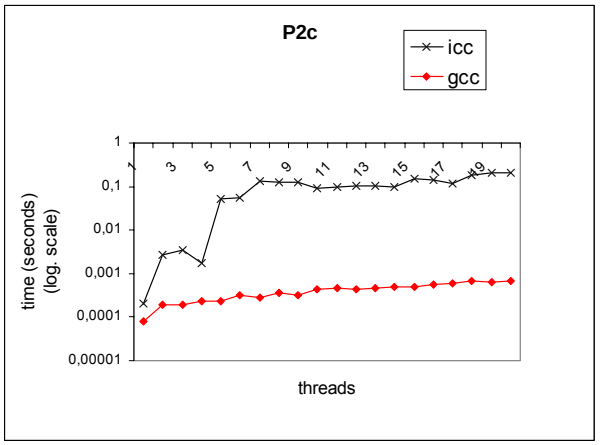
PCB: Benchmarking Routines of the PCE

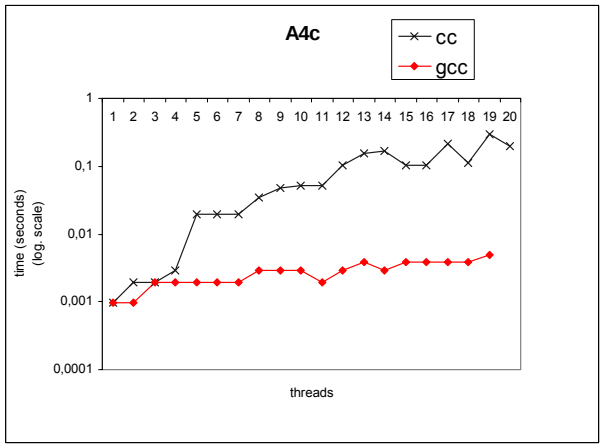
- To test the efficiency of OpenMP primitives for threads creation and management
  - **R-generate**
    - Creating a series of threads with a fixed quantity of work to do per thread
    - To compare the time of creating and managing threads
  - **R-pfor**
    - A simple for loop where there is a signifficant work inside each iteration
    - To compare the time of distributing dynamically a set of homogeneous tasks
  - **R-barriers**
    - A barrier primitive set after a parallel working area
    - To compare the times to perform a global synchronization of all the threads

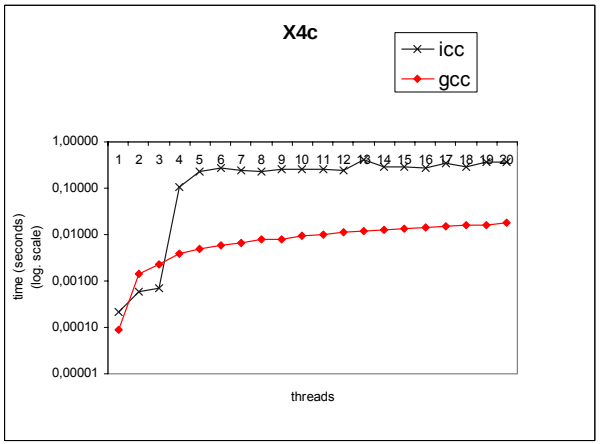
PCB: Benchmarking Routines of the PCE

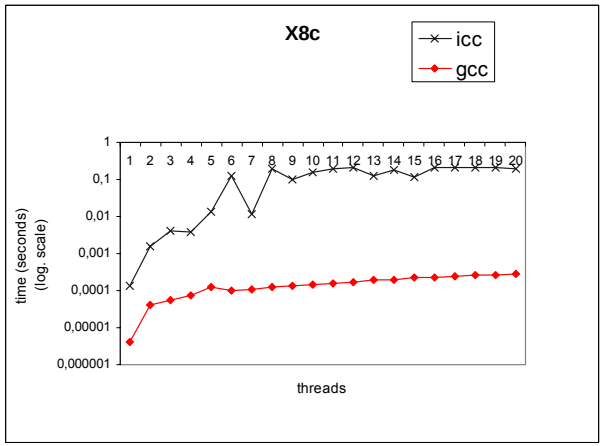
- Experiments: Multicore platforms + compilers:
  - **P2c**
    - Intel Pentium, 2.8 GHz, with 2 cores.
    - Compilers: icc 10.1 and gcc 4.3.2.
  - **A4c**
    - Alpha EV68CB, 1 GHz, with 4 cores.
    - Compilers: cc 6.3 and gcc 4.3.
  - **X4c**
    - Intel Xeon, 3 GHz, with 4 cores.
    - Compilers: icc 10.1 and gcc 4.2.3.
  - **X8c**
    - Intel Xeon, 2 GHz, with 8 cores.
    - Compilers: icc 10.1 and gcc 3.4.6

PCB: Benchmarking Routines of the PCE
R-generate









PCB: Benchmarking Routines of the PCE

R-generate

Execution time model

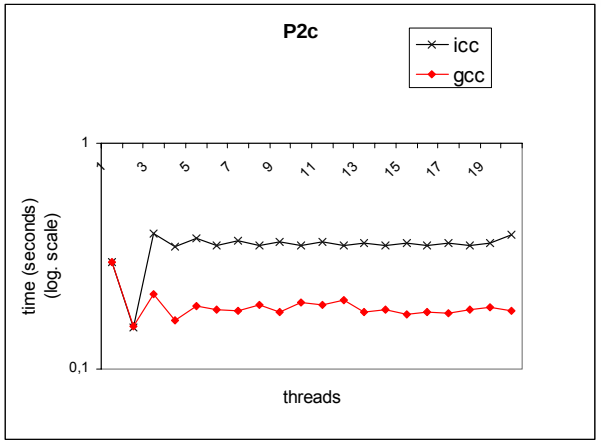
- number of generated threads ≤ number of available cores

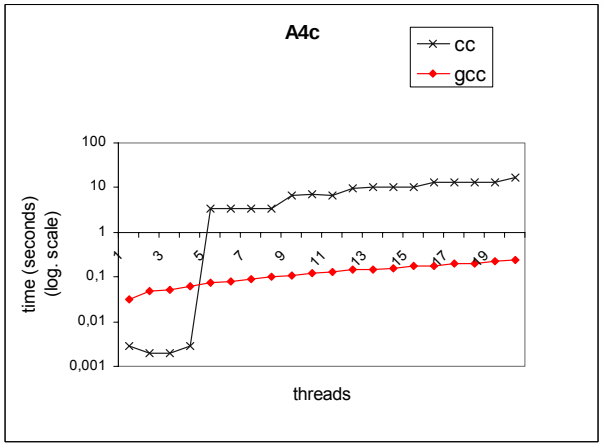
$$T_{R-generate} = PT_{gen} + NT_{work}$$

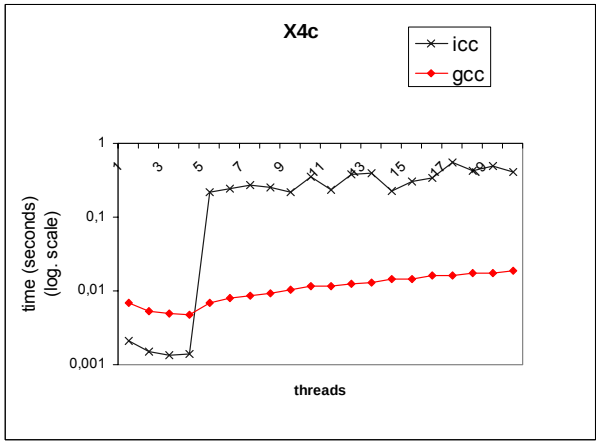
- number of generated threads > number of available cores

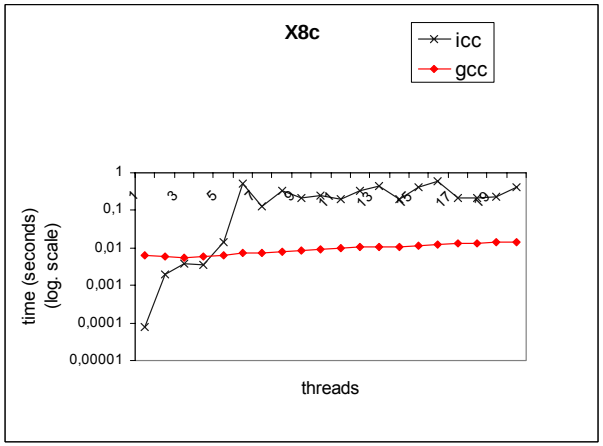
$$T_{R-generate} = PT_{gen} + NT_{work} \frac{P}{C}\left(1 + \frac{T_{swap}}{T_{cpu}} \right)$$

PCB: Benchmarking Routines of the PCE
R-pfor









PCB: Benchmarking Routines of the PCE

R-pfor

Execution time model

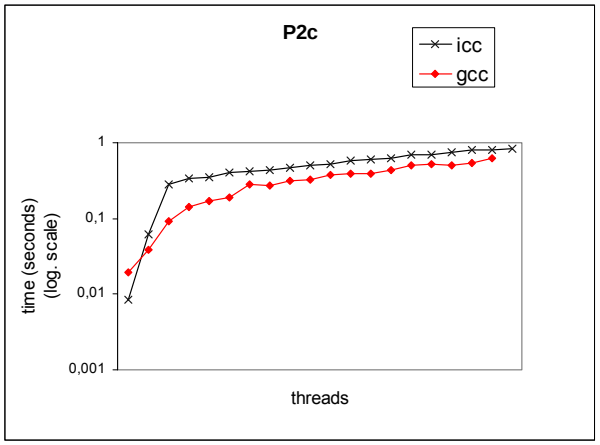
- number of generated threads ≤ number of available cores

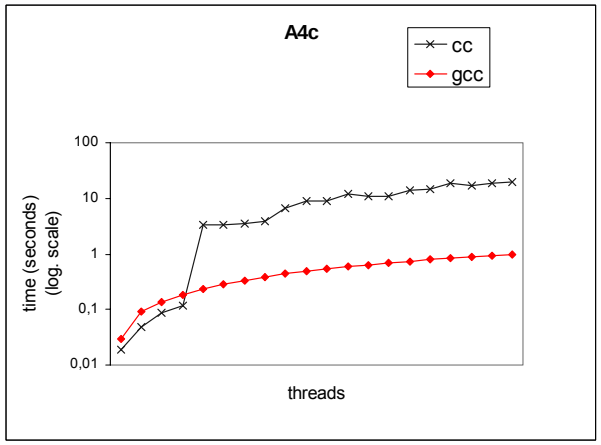
$$T_{R-pfor} = PT_{gen} + \frac{N_T}{P} T_{work}$$

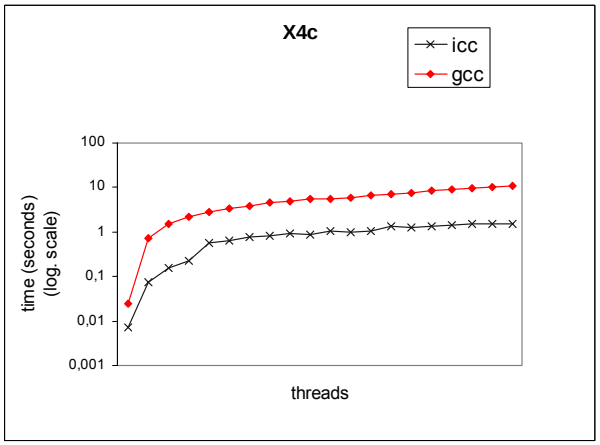
- number of generated threads > number of available cores

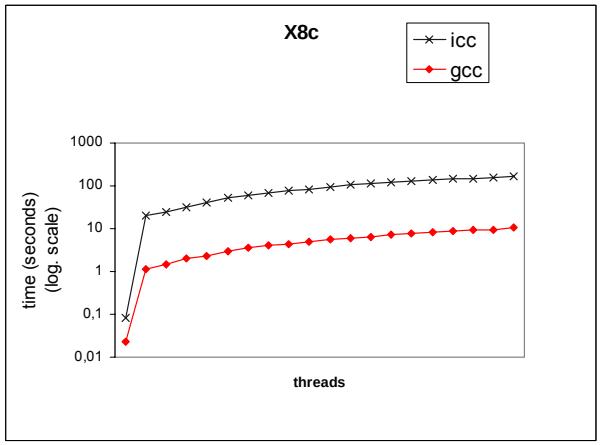
$$T_{R-pfor} = PT_{gen} + \frac{N_T}{C} T_{work} \left(1 + \frac{T_{swap}}{T_{cpu}} \right)$$

PCB: Benchmarking Routines of the PCE
R-barriers









PCB: Benchmarking Routines of the PCE

R-barriers

Execution time model

- number of generated threads ≤ number of available cores

$$T_{R-barriers} = PT_{gen} + NT_{work} + PT_{syn}$$

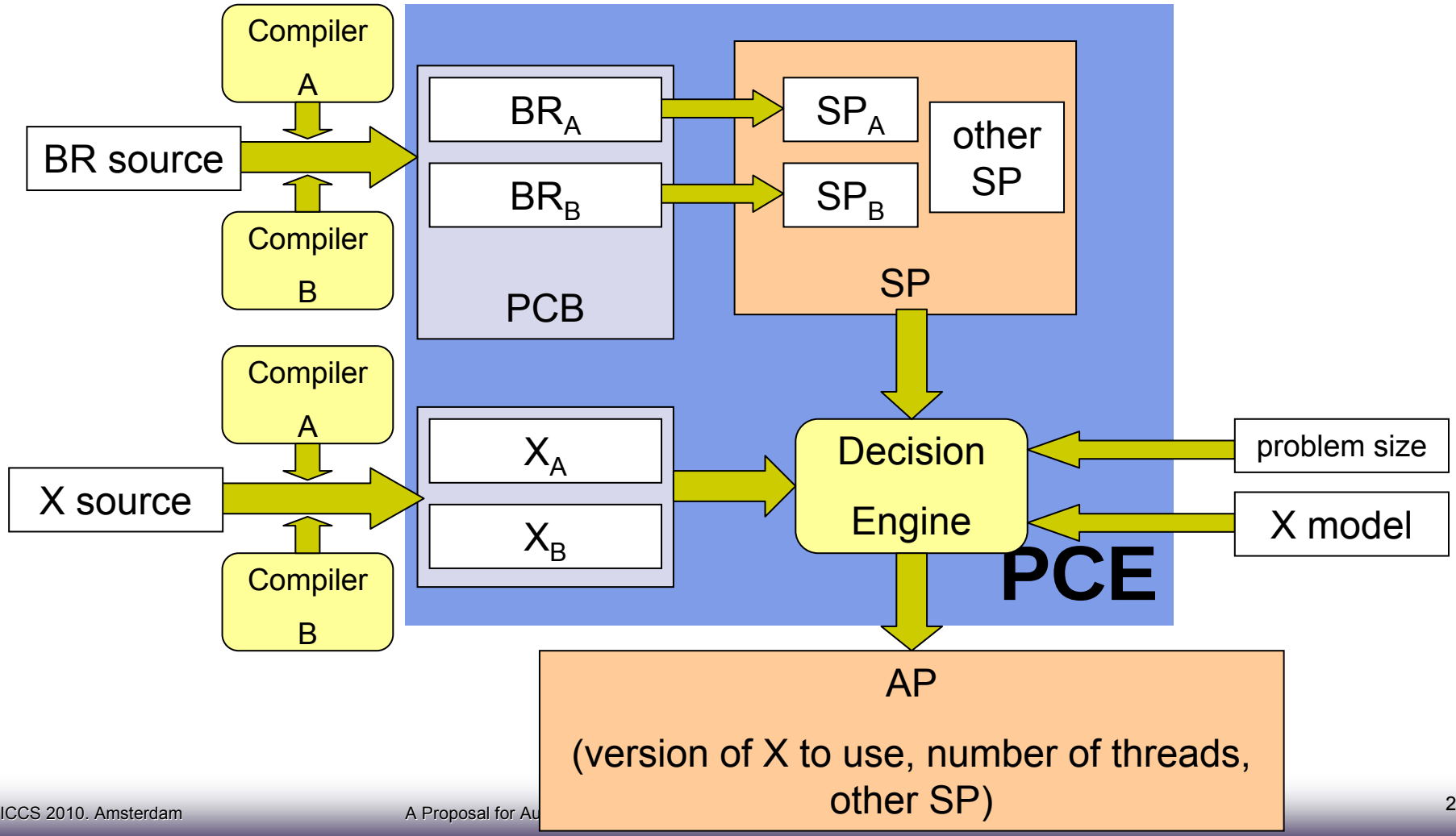
- number of generated threads > number of available cores

$$T_{R-barriers} = PT_{gen} + NT_{work} \frac{P}{C}\left(1 + \frac{T_{swap}}{T_{cpu}}\right) + PT_{syn}$$

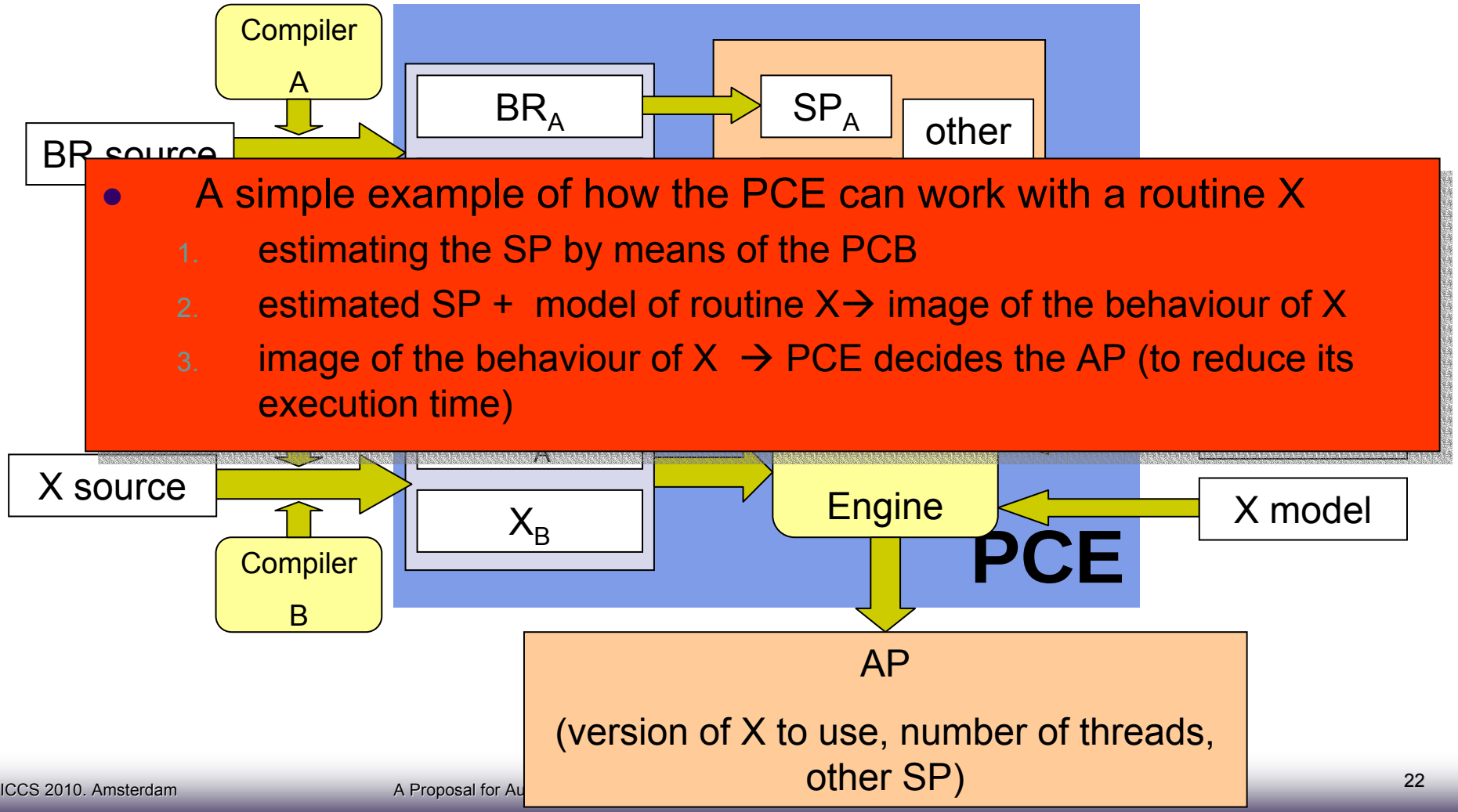
Outline

- Introduction
- Our approach: Poly-Compilation Engine (PCE)
  - PCB: Benchmarking Routines of the PCE
  - Proof of concept of the PCE
- Conclusions

Proof of concept of the PCE



Proof of concept of the PCE routine X



- A simple example of how the PCE can work with a routine X
  1. estimating the SP by means of the PCB
  2. estimated SP + model of routine X→ image of the behaviour of X
  3. image of the behaviour of X → PCE decides the AP (to reduce its execution time)

Proof of concept of the PCE
routine R-strassen
Execution time model

- number of generated threads ≤ number of available cores

$$T_{R-strassen} = PT_{gen} + 7\frac{2\left(\frac{n}{2}\right)^3}{P}T_{mult} + \frac{18n^2}{4}T_{add}$$

$$T_{R-strassen} = P_1P_2T_{gen} + 49\frac{2\left(\frac{n}{2^2}\right)^3}{P_2P_1}T_{mult} + 7\frac{\frac{18n^2}{4^2}}{P_1}T_{add} + 18\frac{n^2}{4}T_{add}$$

- number of generated threads > number of available cores

$$T_{R-strassen} = PT_{gen} + 7\frac{2\left(\frac{n}{2}\right)^3}{C}T_{mult}\left(1 + \frac{T_{sw}}{T_{cpu}}\right) + \frac{18n^2}{4}T_{add}$$

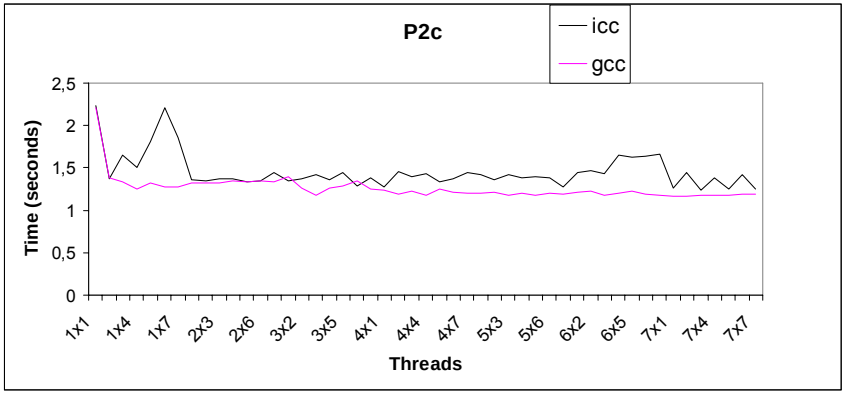
$$T_{R-strassen} = P_1P_2T_{gen} + 49\frac{2\left(\frac{n}{2^2}\right)^3}{C}T_{mult}\left(1 + \frac{T_{sw}}{T_{cpu}}\right) + 7\frac{\frac{18n^2}{4^2}}{min(P_1, C)}T_{add}\left(1 + \frac{T_{sw}}{T_{cpu}}\right) + 18\frac{n^2}{4}T_{add}$$

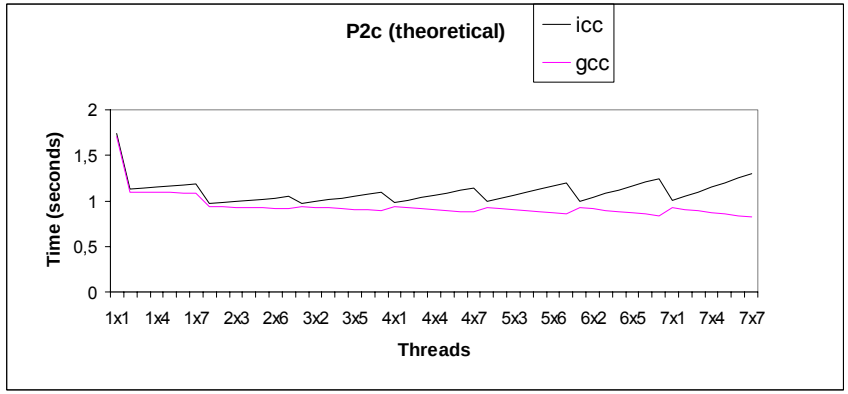
Proof of concept of the PCE
routine R-strassen
SP values

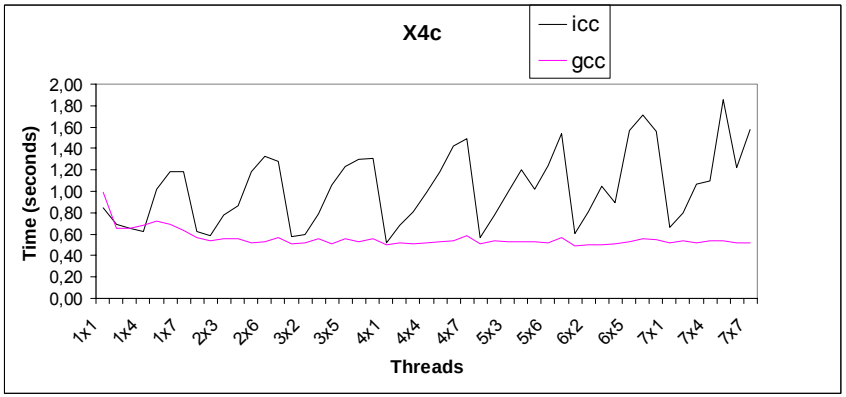
	P2c		X4c		A4c		X8c	
	icc	gcc	icc	gcc	cc	gcc	icc	gcc
T_{gen} (μs)	75	25	75	25	75	25	75	25
T_{sw}/T_{cpu}	$2 + 0.01P$	$7 - 0.01P$	$0.9 + 0.3P$	$0.9 + 0.01P$	$0.8 + 0.2P$	$0.8 + 0.02P$	$6 + 0.05P$	$0.5 + 0.01P$
T_{add} (ns)	$20 + 0.05P$	20	$23 + 0.3P$	$30 - 0.3P$	$40 + P$	$40 - 0.1P$	10	10
T_{mul} (ps)	$400 + 100P$	$400 + 0.1P$	$140 + 10P$	$140 - P$	60	$60 - 0.5P$	100	100

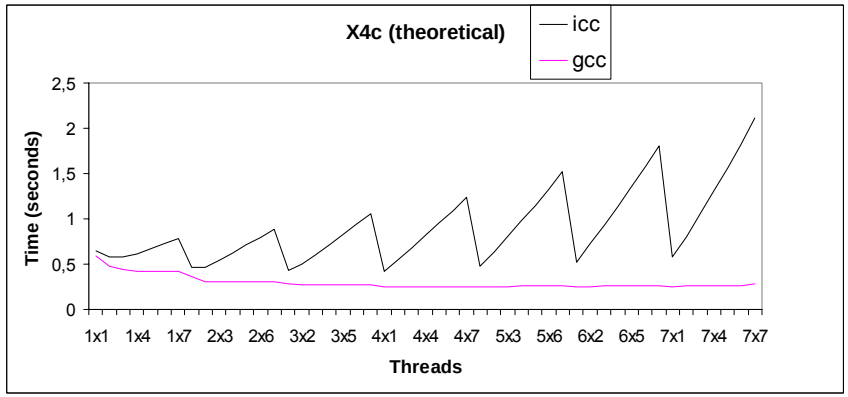
Proof of concept of the PCE **routine R-strassen**

Experimental Vs. Theoretical times



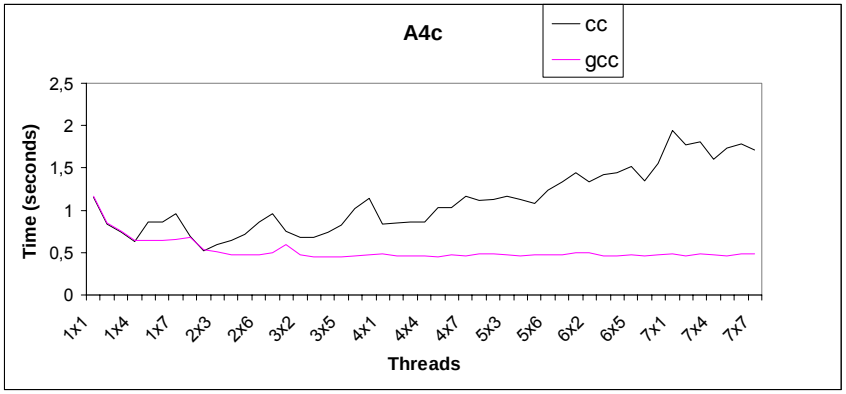


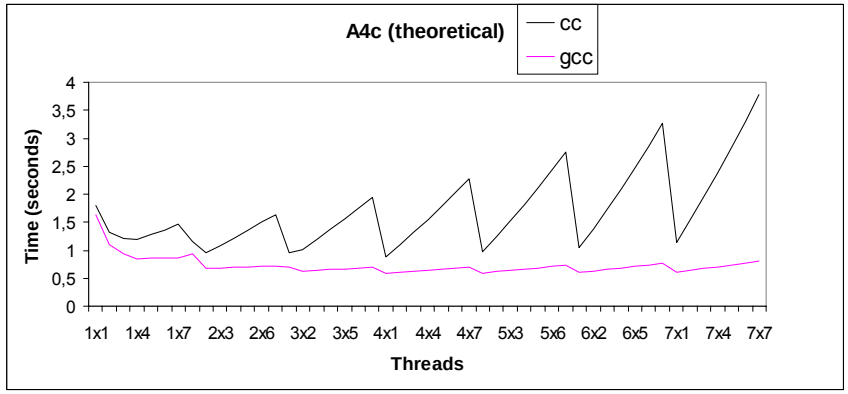


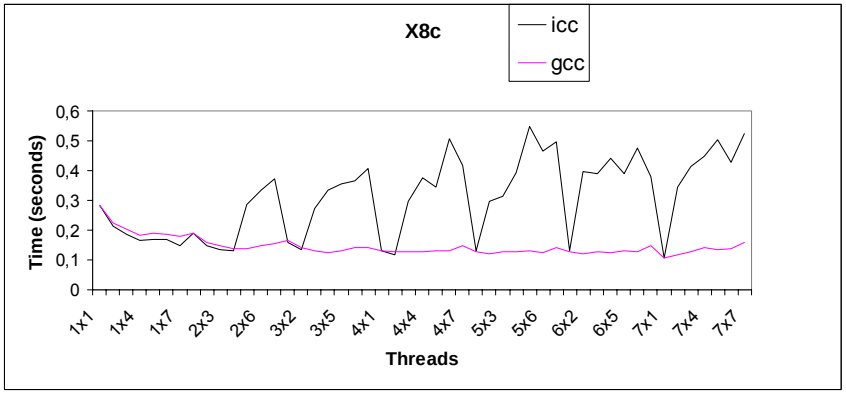


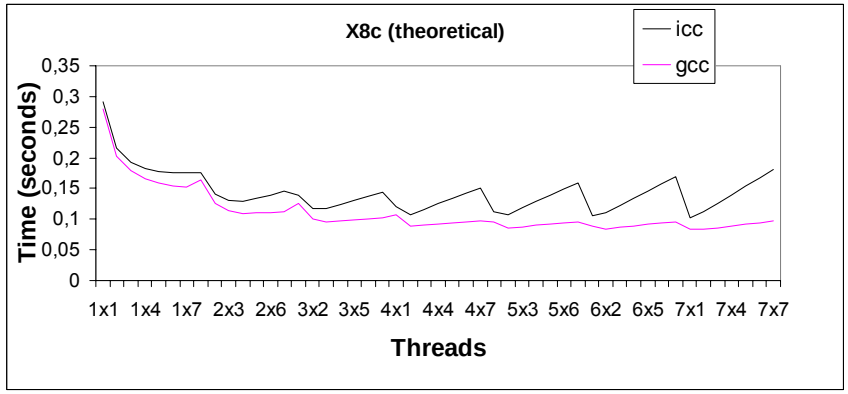
Proof of concept of the PCE **routine R-strassen**

Experimental Vs. Theoretical times









Proof of concept of the PCE **routine R-strassen** taking decisions: AP values

	P2c	X4c	A4c	X8c
Compiled version	gcc	gcc	gcc	gcc
Number of threads for the 1st level	7	4	4	7
Number of threads for the 2nd level	7	1	1	2

Algorithmic Parameters values, for the R-strassen routine, taken by the Poly-Compilation Engine for different platforms and compilers (Problem size=1000).

Proof of concept of the PCE
routine R-strassen
executing the routine

	P2c	X4c	A4c	X8c
PCE	1.19	0.50	0.49	0.16
ORA	1.17	0.49	0.45	0.11
HW	1.37	0.55	0.65	0.12
SW	1.22	1.31	1.20	0.32

Execution times of the R-strassen routine obtained with different Algorithmic Parameters values sets (times in seconds). Problem size=100

Outline

- Introduction
- Our approach: Poly-Compilation Engine (PCE)
  - PCB: Benchmarking Routines of the PCE
  - Proof of concept of the PCE
- Conclusions

Conclusions

- A good choice of compiler in a multicore contributes to accelerating the solution of problems
- Best compiler depends on: the type of routine, the number of threads, the problem size,...
- Working on the design and implementation of a complete Poly-Compilation Engine (PCE).
  - Calculates the basic System Parameters (SP) for platform-compiler.
  - Selects the Algorithmic Parameters (the compiled version and the number of threads), by means of the theoretical model of the execution time of the routine with the SP measurement
- The behaviour of a PCE prototype has been shown with a routine
- Similar studies with other linear algebra routines are being performed.