PetaQCD: Code generators

- Code generators : design, validation
- Code analysis: how to evaluate code quality

F. Bodin (Caps-Entreprise), D. Barthou, C. Bastoul, H.-P. Charles, A. Cohen, S. Touati (Alchemy/LRI/Inria/Paris-Sud and Prism/UVSQ)

What is LQCD computation for us today?

Para (ANR) project:

- ETMC/HMC code = irregular at first sight (complex address computations due to memory layout of 4D-lattice).
- Rewritten into a 4D loop scanning the lattice sites and explicit 4D memory access.
- A lot of vector computations
- 1600 Flops / site

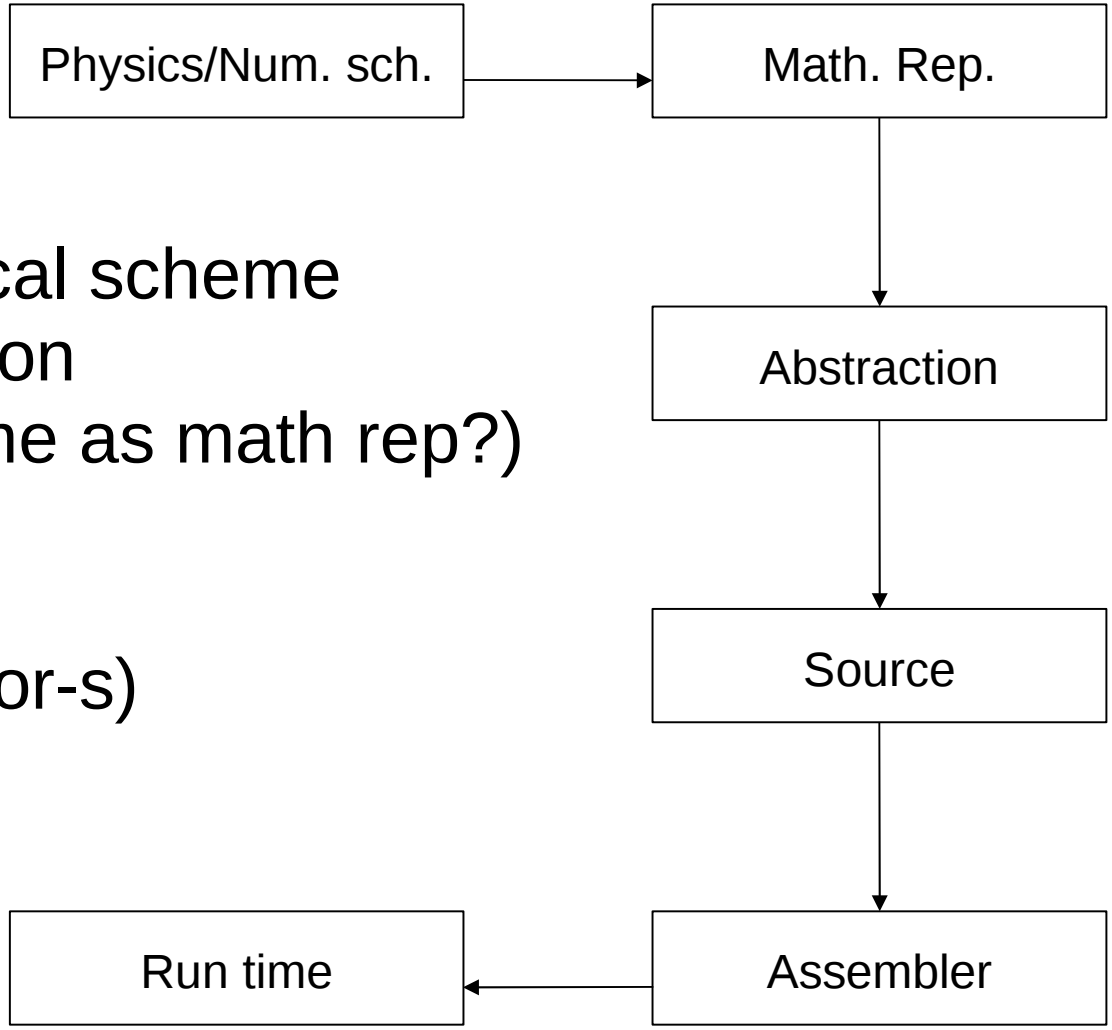
Perfect candidate for loop parallelization tools (polyhedric representation ... by construction).

But the code is **communication-bound**. (evaluation without gauge matrix reuse)

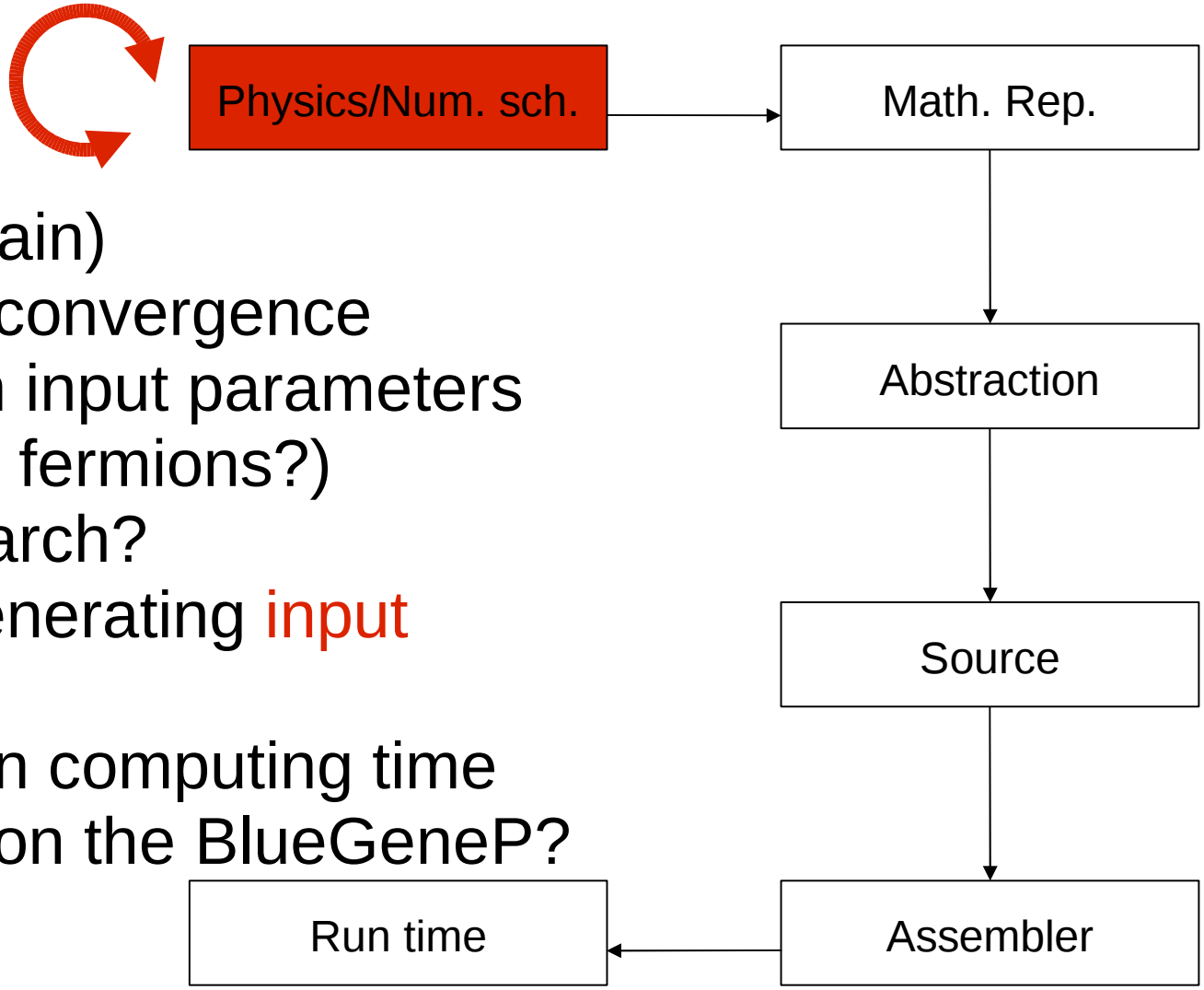
Complexity of processors and communication networks => no static cost model possible => try and test/iterative approach

Different levels

- physics – numerical scheme
- math representation
- "abstraction" (same as math rep?)
- source code
- assembly code
- run time (processor-s)



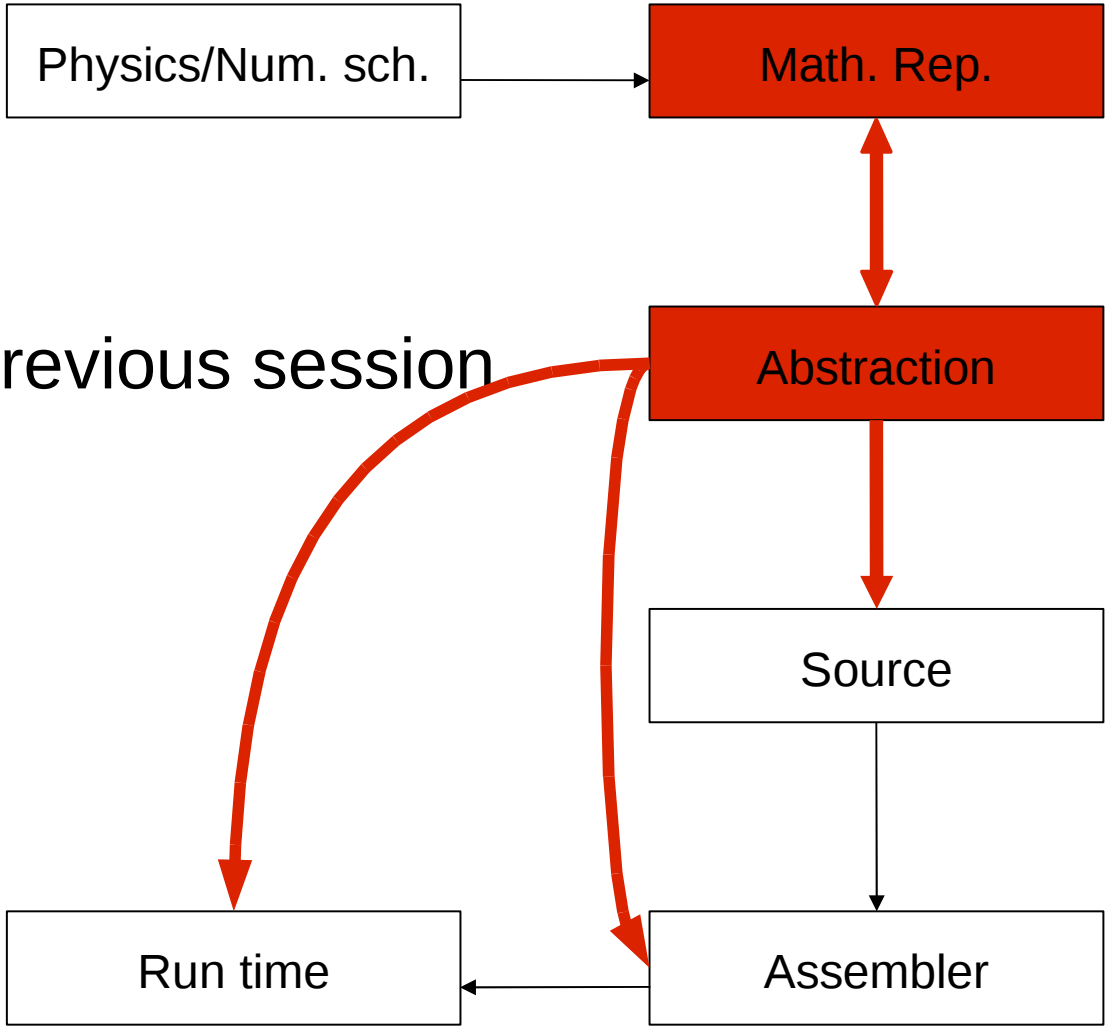
Physics/Numerical Scheme



- by hand (brain)
- analysis of convergence
- depends on input parameters (heavy/light fermions?)
- iterative search?
- script for generating input parameters
- expensive in computing time
- to be done on the BlueGeneP?

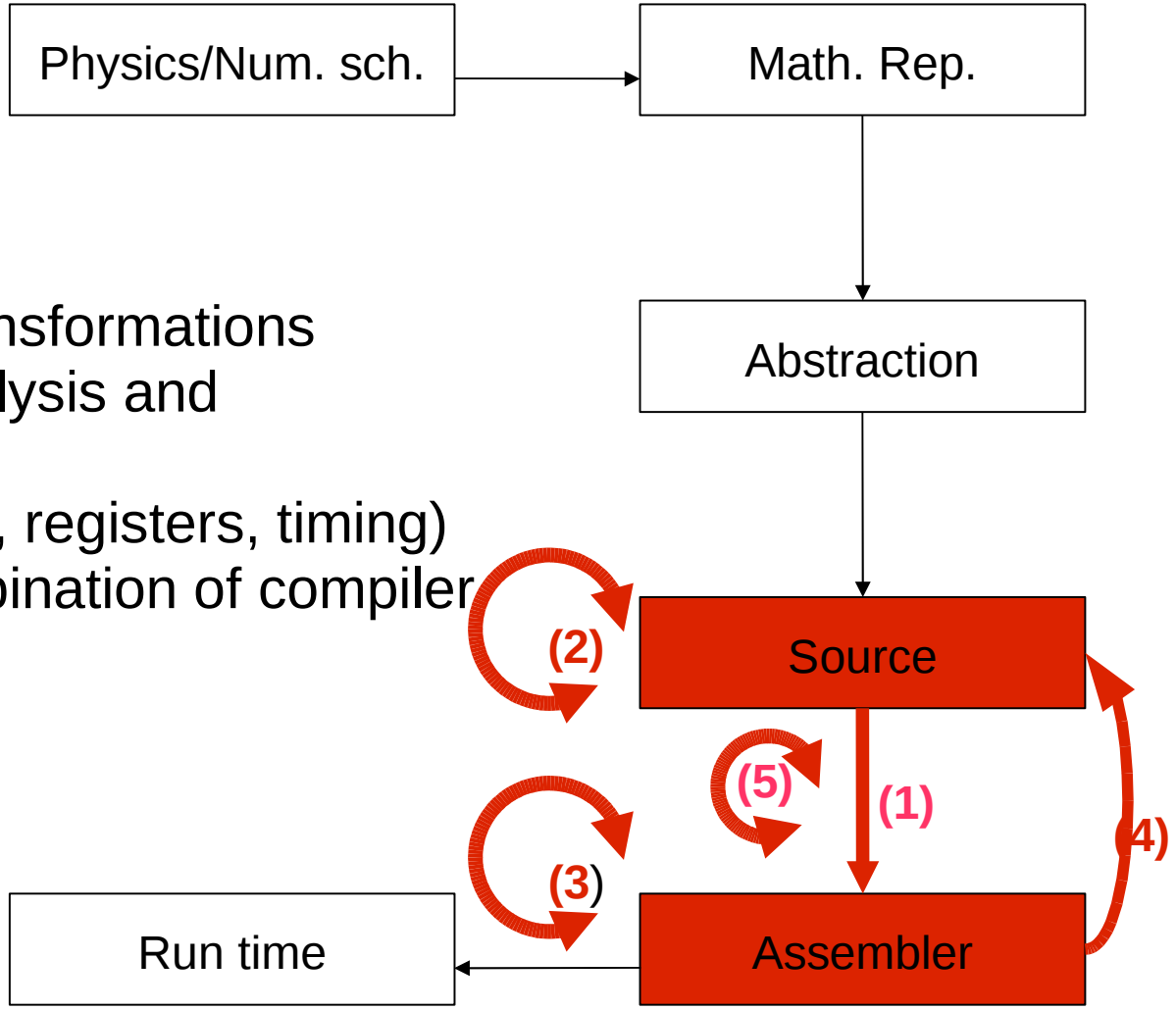
Math. Rep./Abstraction

- discussed in the previous session
- same level?



Source – assembler

(1) Compiler
(2) Source-to-source transformations
(3) Assembler static analysis and transformations
(4) Feedback (code size, registers, timing)
(5) Search for best combination of compiler arguments

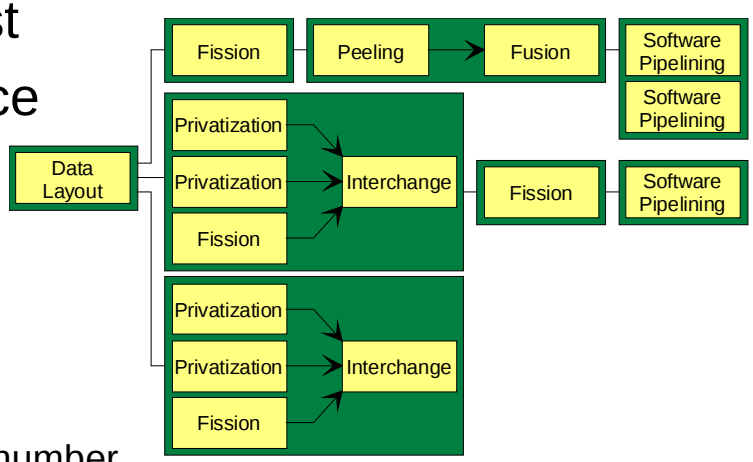


Manual Optimizations (S.Girbal, Alchemy)

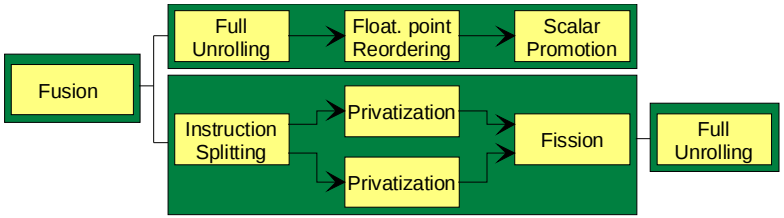
- Long sequences of transformations
- Transformations often hits the same loop nest
- Some “enabling” phases degrade performance

	peak	manual
swim	1,00	1,61
galgel	1,04	1,39
applu	1,47	2,18
apsi	1,07	1,23

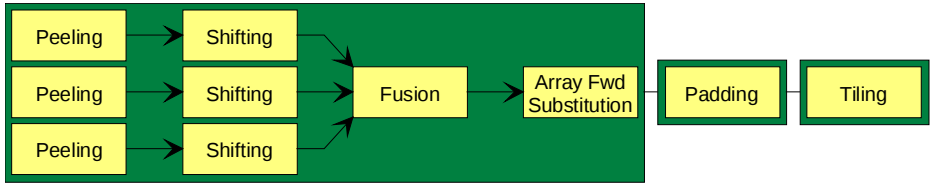
- Mainly regular loop nests
  - High number of loop nests
  - Low number of conditionals
- Performance loop nests
  - High variance in instruction number
  - Small loop depth



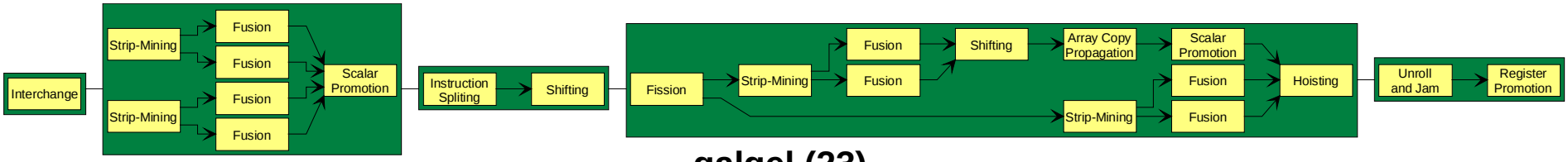
apsi (16)



applu (9)



swim (10)

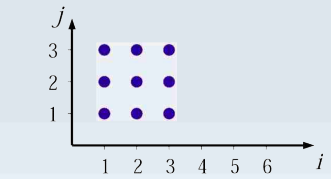


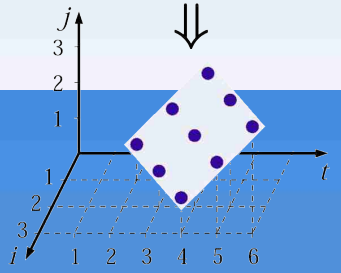
galgel (23)

High Level Program Optimization (C. Bastoul, Alchemy)

Why ? Extract high level properties to restructure/optimize programs
How ? Switch to an algebraic representation, *the Polyhedral Model*

```
for (i=1;i<=3;i++)
  for (j=1;j<=3;j++)
    A[i+j] = ...
```






```
for (t=2;t<=6;t++)
  for (i=max(1,t-3);i<=min(t-1,3);i++)
    A[t] = ...
```

A Three Step Approach

Analysis: putting programs in the model

WRAP-IT, GRAPHITE tools
Pop et al. *GRAPHITE : Loop optimizations based on the polyhedral model for GCC,* GCC Developper's Summit 2006
Girbal et al. *Putting Polyhedral Loop Transformation to Work*, LCPC 2003

Transformation: finding optimizing ones

Letsee, Polydeps, Chunky tools
Pouchet et al. *Iterative Optimization in the Polyhedral Model*, PLDI 2008, CGO 2007
Vasilache et al. *Violated Dependence Analysis*, ICS 06
Cohen et al. *Facilitating the Search for Compositions of Program Transformations*, ICS 2005

Generation: generating target programs

CLooG tool
Vasilache et al. *Polyhedral Code Generation in the Real World*, CC 2006
Bastoul *Code Generation in the Polyhedral Model Is Easier Than You Think*, PACT 2004

Lessons from the PARA Project

Analysis: Fighting Irregularity

- LQCD codes are not (fully) regular
- We develop extensions to the Polyhedral Model to manipulate *full functions*

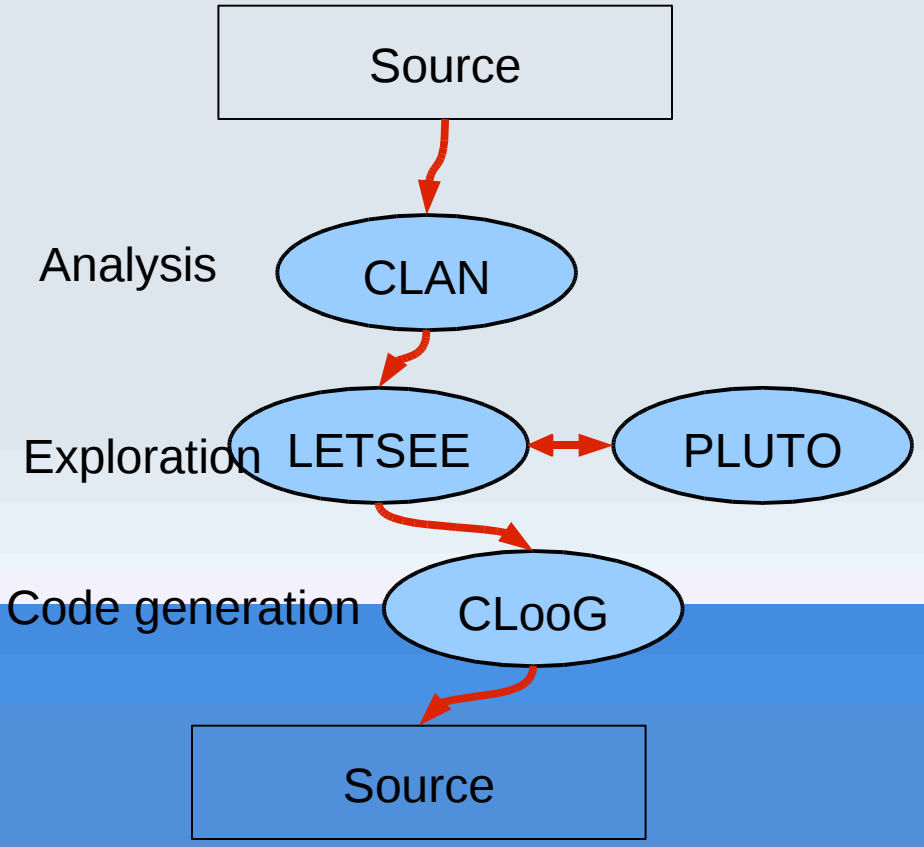
Transformations: Pushing Automated Version Generation

- Iterative Optimization in the Polyhedral Model prove to be qu successful
- We need extensions for parallelism
- Performance evaluation is critical

Code Generation: Targeting New Architectures

- A CUDA backend for ClooG is under active development
- Cell, Larabee should be investigated

Alchemy tools (Bastoul, Cohen et al.)



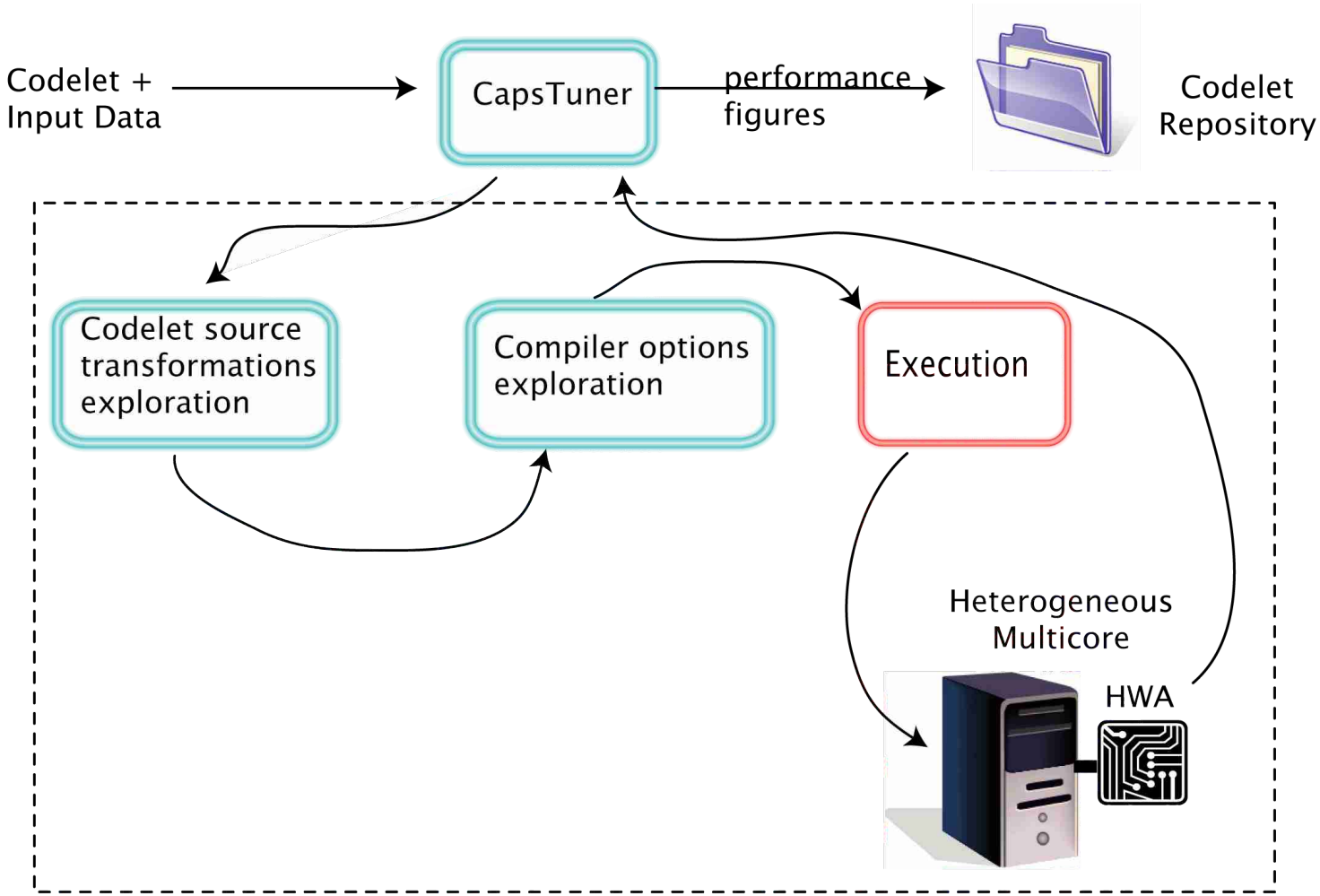
- LETSEE = affine by statement
- PLUTO adds tiling
- Generated source = OpenMP, no MPI (yet?)
- Expected = backend for CUDA, Cell?
- Former Graphite environment integrated in the 4.4 gcc version, parallelism extraction expected in gcc 4.5

Codelet Tuning

Codelet Tuning (1)

- Based on CAPSTuner Technology
  - Platform independent
- Iterative compilation technique to explore the optimization space
  - Explore source code transformations
    - Via a set of code transformation directives such as unroll-and-jam
  - Explore compiler options
  - Store the performance data in a well defined repository

Code Tuning (2)



MAQAO (D. Barthou et al., Prism)

Objective of the tool

- Identify performance bottleneck of one thread code
- Propose performance tuning hints

Identifying single thread performance issues

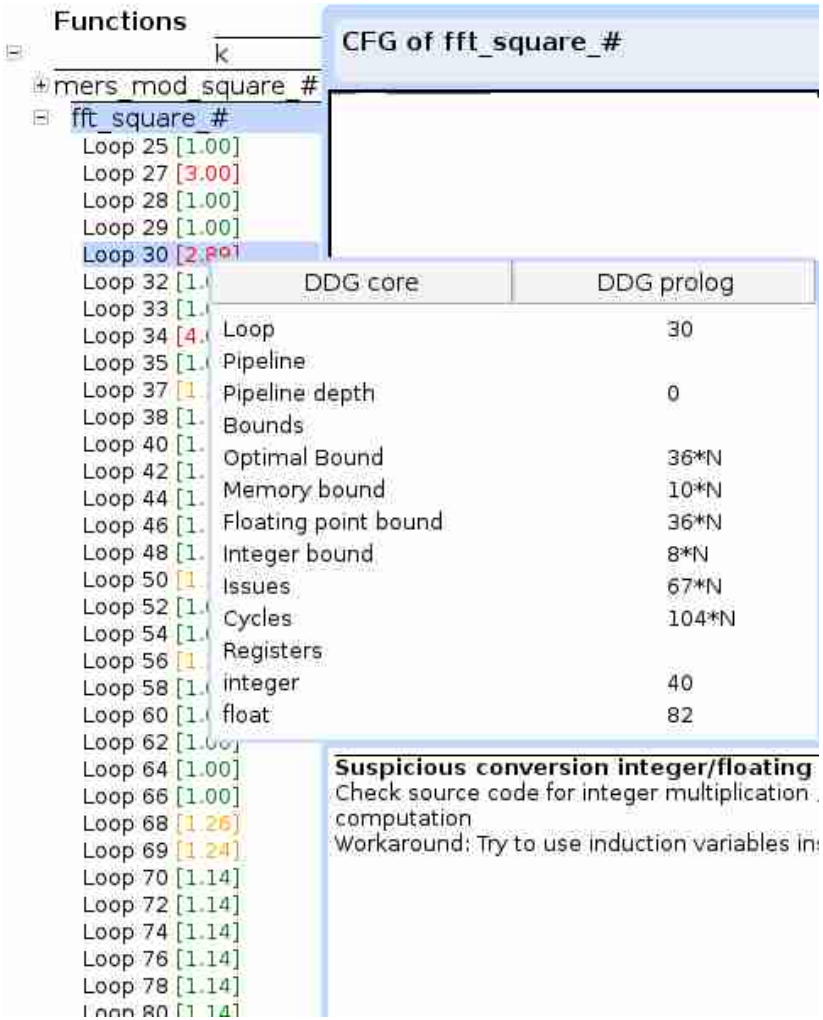
- Collect static information from assembly codes
  - Detect inefficient code patterns
  - Static performance evaluation for inner loops
- Collect value profiling information
  - Function parameters,
  - Prefetch values,
  - Loop trip count distribution...

Estimate potential performance gain

- For each inner loop
- For each function hotpath

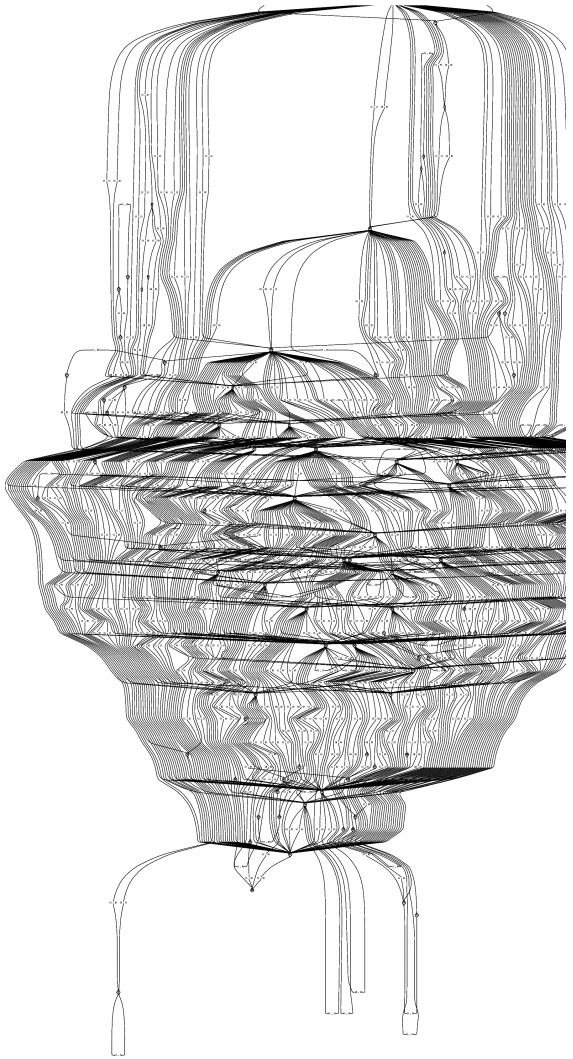
Find opportunities for specialization

- Loop trip counts
- Function parameters



SIRA (S. Touati, Prism)

- Single loop
- Evaluates memory (register) usage
- Minimun required
- Maximum required for any – possibly parallel – operations schedule.
- Input = DAG or LoopDAG (LDDG, Loop Data Dependency Graph)
- Register usage analysis
- Register usage optimization

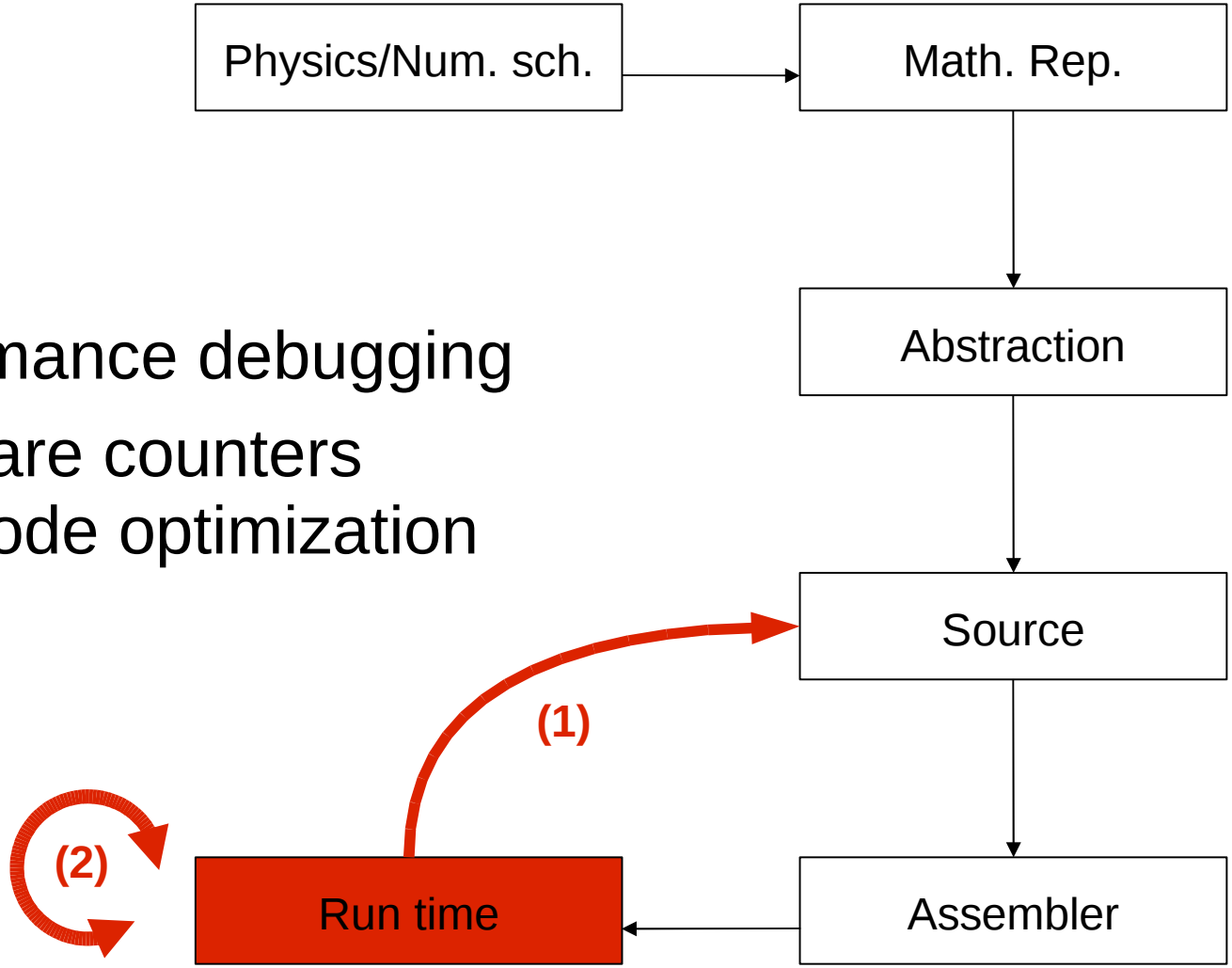


Run time

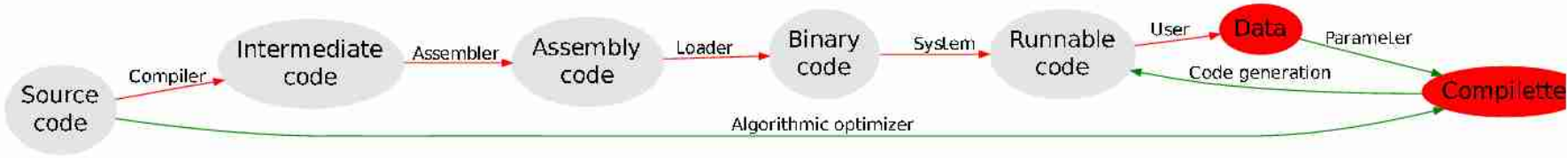
(1)Profiling

- Performance debugging
- Hardware counters

(2)Dynamic code optimization



Compilettes : a dynamic code generator



Unusual code generation at call-time. (not compile time like C, not run-time like java)

Input High level algorithm description

Output Highly optimized binary code

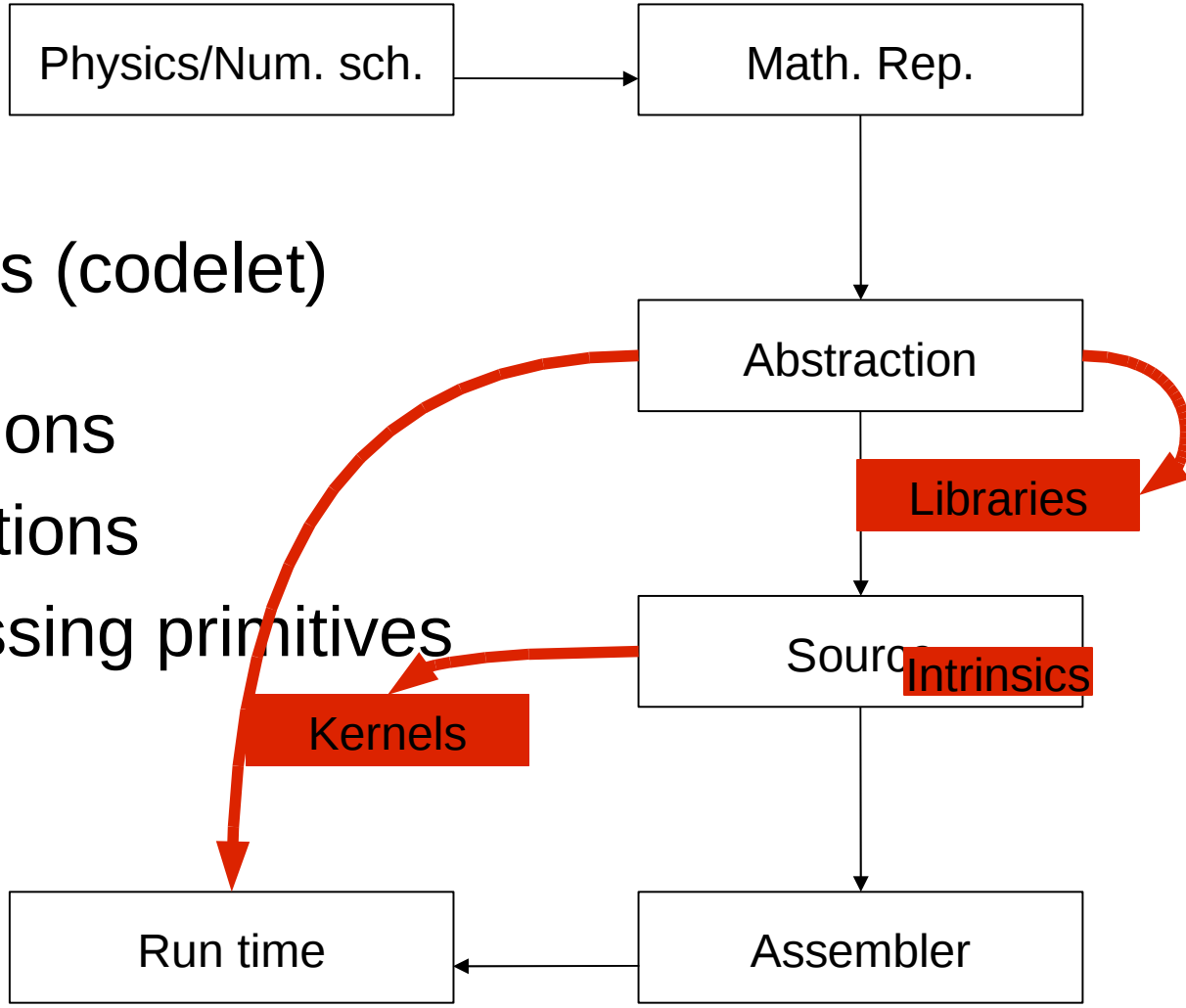
When at call time (when data values and memory aligments are known)

Expected results :

- Correct use of the dual floating point special instructions on blugene (the IBM compiler fails)
- Handling memory alignments
- Others parameters to be discovered

Other levels

- Computing kernels (codelet)
- Libraries
- Parallel constructions
  - Vector operations
  - Message passing primitives
- PRAGMA
  - OpenMP
- Intrinsics



Xlanguage (Prism)

Exploring many optimizations

- Specify optimization space to explore with pragmas, for a given code fragment

  [Xlanguage, LCPC05, Donadio *et al*.]

- Source to source transformations: unroll, interchange, tiling, ...
  - Can be user-defined, application dependent
  - Optimization space user-defined
- Generate multiple kernels versions
- (combined with MAQAO)


hmpp
hybrid multicore parallel programming

Simple C Example

To be executed on the HWA

```
#include <stdio.h>
#include <stdlib.h>

#pragma hmpp simple codelet, args[1].io=out
void simplefunc(int n, float v1[n], float v2[n], float v3[n], float alpha)
{
  int i;
  for (i = 0 ; i< n ; i++) {
    v1[i] = v2[i] * v3[i] + alpha;
  }
}
int main(int argc, char **argv) {
  unsigned int n = 400;
  float t1[400], t2[400], t3[400];
  float alpha = 1.56;
  unsigned int  j, seed = 2;
/* Initialization of input data*/
/* . . . */

#pragma hmpp simple callsite
  simplefunc(n,t1,t2,t3,alpha);

  printf("%f %f (...) %f %f \n", t1[0], t1[1], t1[n-2], t1[n-1]);
  return 0;
}
```

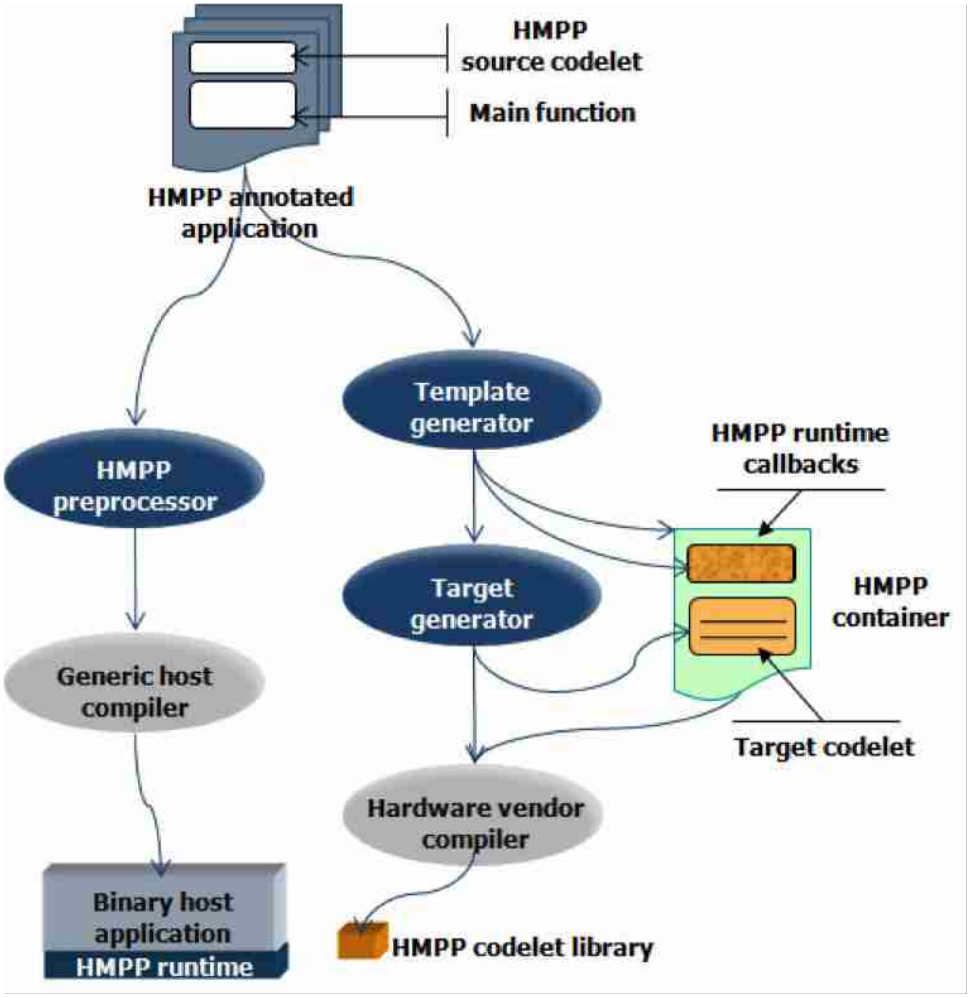
codelet / callsite directive set

Codelet Generation

Objectives

- Allow to transparently use HWA
  - From C or Fortran, Java to CUDA, Brook, ...
- Allow for code tuning at source code level
  - Directives based approach

Code Generation Flow



Codelet Generation

- C, Java or Fortran source code input
  - HWA oriented subset of the languages
- Set of directives to
  - Optimize target codelet generation
  - Express parallelism expression
- Make code tuning easier
- Generated code can also be tuned

Loop Parallelization

- Force or prevent the parallelization of loops
- Help defining kernels in a codelet

```
#pragma hmppcg parallel
for (i=0; i < n; i++) {
#pragma hmppcg noParallel
    for (j=0; j < n; j++) {
          D[i][j] = A[i][j] * E[3][j];
     }
}
```

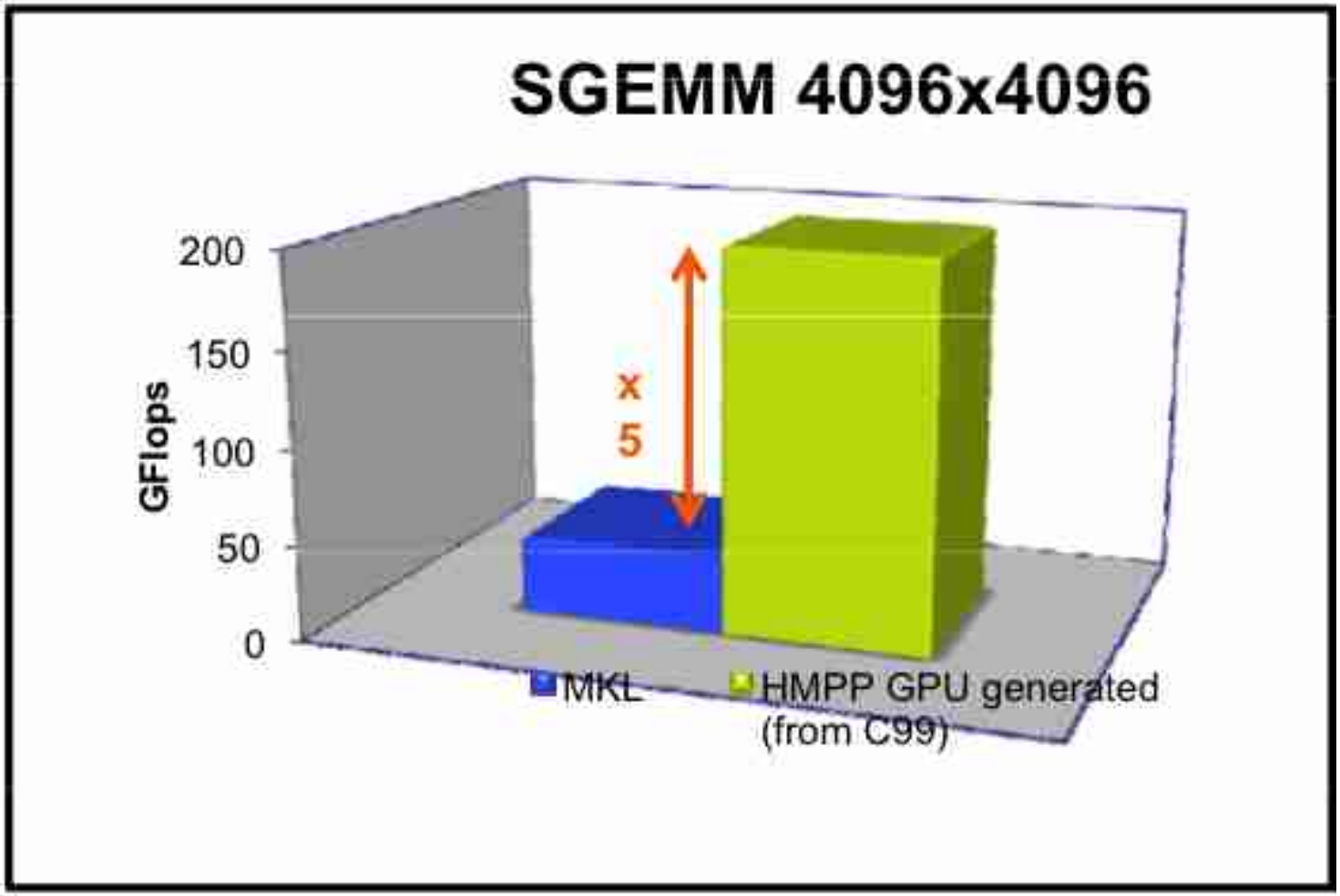
Input C Code Example 1

```
typedef struct{    float r, i;} Complex;
#pragma hmpp convolution2d codelet, args[data; opx].io=in, args[convr].io=out, target=CUDA
void convolution2d( Complex *data, int nx, int ny,  Complex *opx,
                    int oplx, int oply, Complex *convr) {
int hoplx = (oplx+1)/2;  int hoply = (oply+1)/2;
int iy, ix;
#pragma hmppcg parallel
for (iy = 0; iy < ny; iy++) {
#pragma hmppcg parallel
  for (ix = 0; ix < nx; ix++) {
    float dumr =0.0, dumi = 0.0;      int ky;
    for(ky = -(oply - hoply - 1); ky <= hoply; ky++) {
      int kx;
      for(kx = -(oplx - hoplx - 1); kx <= hoplx; kx++){
        int dx = min( max(ix+kx, 0), (nx - 1) );
        int dy = min( max(iy+ky, 0), (ny - 1) );
        dumr += data[dy * nx + dx].r * opx[(hoply - ky) * oplx + (hoplx - kx)].r;
        dumr -= data[dy * nx + dx].i * opx[(hoply - ky) * oplx + (hoplx - kx)].i;
        dumi += data[dy * nx + dx].r * opx[(hoply - ky) * oplx + (hoplx - kx)].i;
        dumi += data[dy * nx + dx].i * opx[(hoply - ky) * oplx + (hoplx - kx)].r;
       }
     }
     convr[iy*nx+ix].r = dumr; convr[iy*nx+ix].i = dumi;
   }
  }
}
```

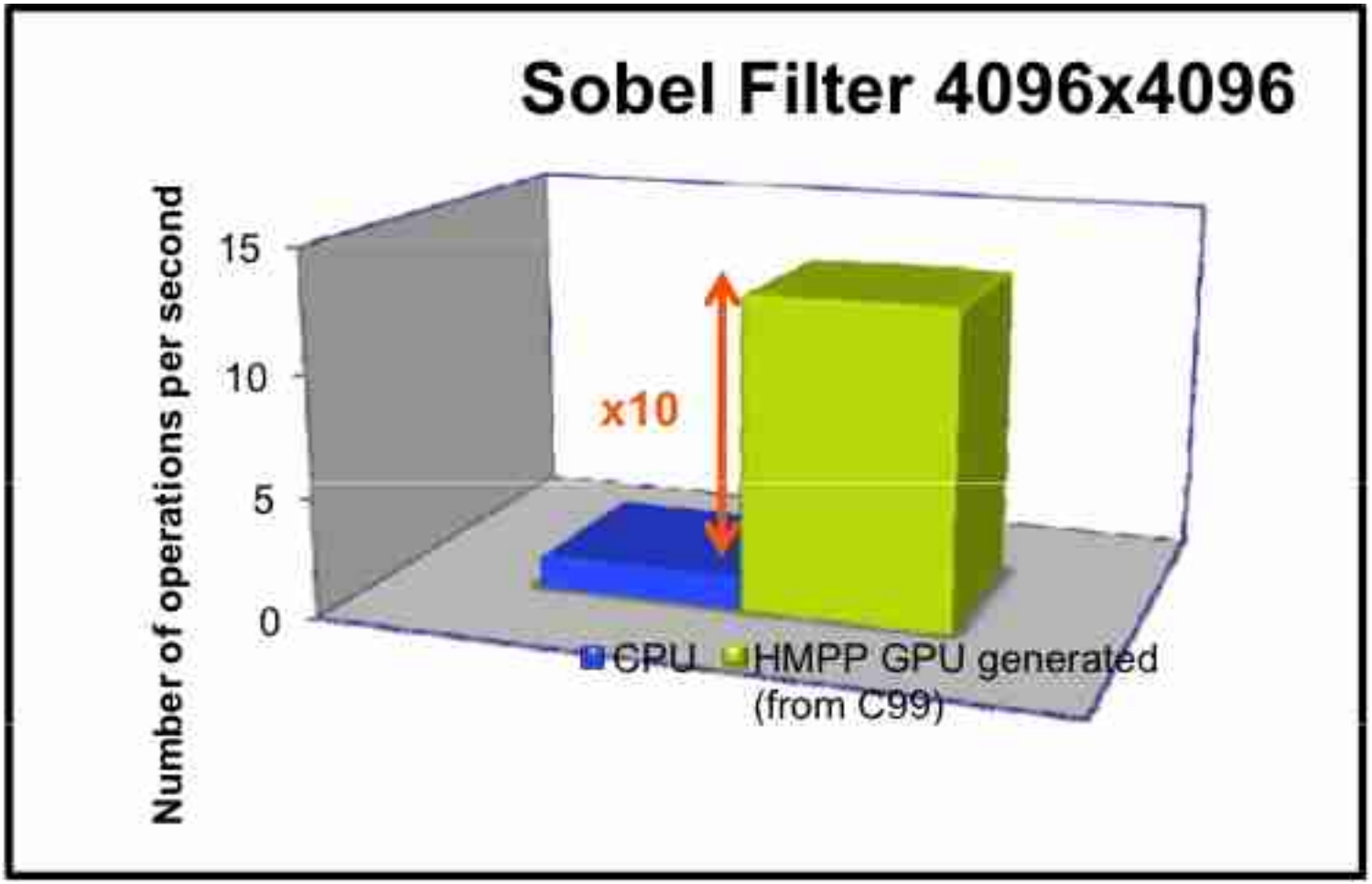
Input Fortran Code Example 2

```
!$HMPP sgemm3 codelet, target=CUDA, args[vout].io=inout
SUBROUTINE sgemm(m,n,k2,alpha,vin1,vin2,beta,vout)
INTEGER, INTENT(IN)    :: m,n,k2
REAL,   INTENT(IN)    :: alpha,beta
REAL,    INTENT(IN)    :: vin1(n,n), vin2(n,n)
REAL,    INTENT(INOUT) :: vout(n,n)
REAL     :: prod
INTEGER  :: i,j,k
!$HMPPCG unroll(8), jam(2), noremainder
!$HMPPCG parallel
DO j=1,n
    !$HMPPCG unroll(8), splitted, noremainder
    !$HMPPCG parallel
    DO i=1,n
        prod = 0.0
        DO k=1,n
          prod = prod + vin1(i,k) * vin2(k,j)
        ENDDO
        vout(i,j) = alpha * prod + beta * vout(i,j) ;
     END DO
END DO
END SUBROUTINE sgemm
```

MxM Performance



Performance Examples



Questions

- Importance of starting from a high level representation to have a larger search space.
- Two extremes: Parallel 4D loop over 1600 FLOPS <-> Sequence of 1600 vector operations and all combinations between both.
- Huge combinatorics of transformations
  - machine learning/genetic algorithms used.
  - Huge space of configurations (MonteCarlo search)?
  - Are we exploring in the right search space?
- How to cope with communications issue?
  - Point to point communcation
  - Desynchronization (num scheme issue)
  - Work on num. scheme. (which alternatives)?


hmpp
hybrid multicore parallel programming

Simple C Example

To be executed on the HWA

```
#include <stdio.h>
#include <stdlib.h>

#pragma hmpp simple codelet, args[1].io=out
void simplefunc(int n, float v1[n], float v2[n], float v3[n], float alpha)
{
  int i;
  for (i = 0 ; i< n ; i++) {
    v1[i] = v2[i] * v3[i] + alpha;
  }
}
int main(int argc, char **argv) {
  unsigned int n = 400;
  float t1[400], t2[400], t3[400];
  float alpha = 1.56;
  unsigned int  j, seed = 2;
/* Initialization of input data*/
/* . . . */

#pragma hmpp simple callsite
  simplefunc(n,t1,t2,t3,alpha);

  printf("%f %f (...) %f %f \n", t1[0], t1[1], t1[n-2], t1[n-1]);
  return 0;
}
```

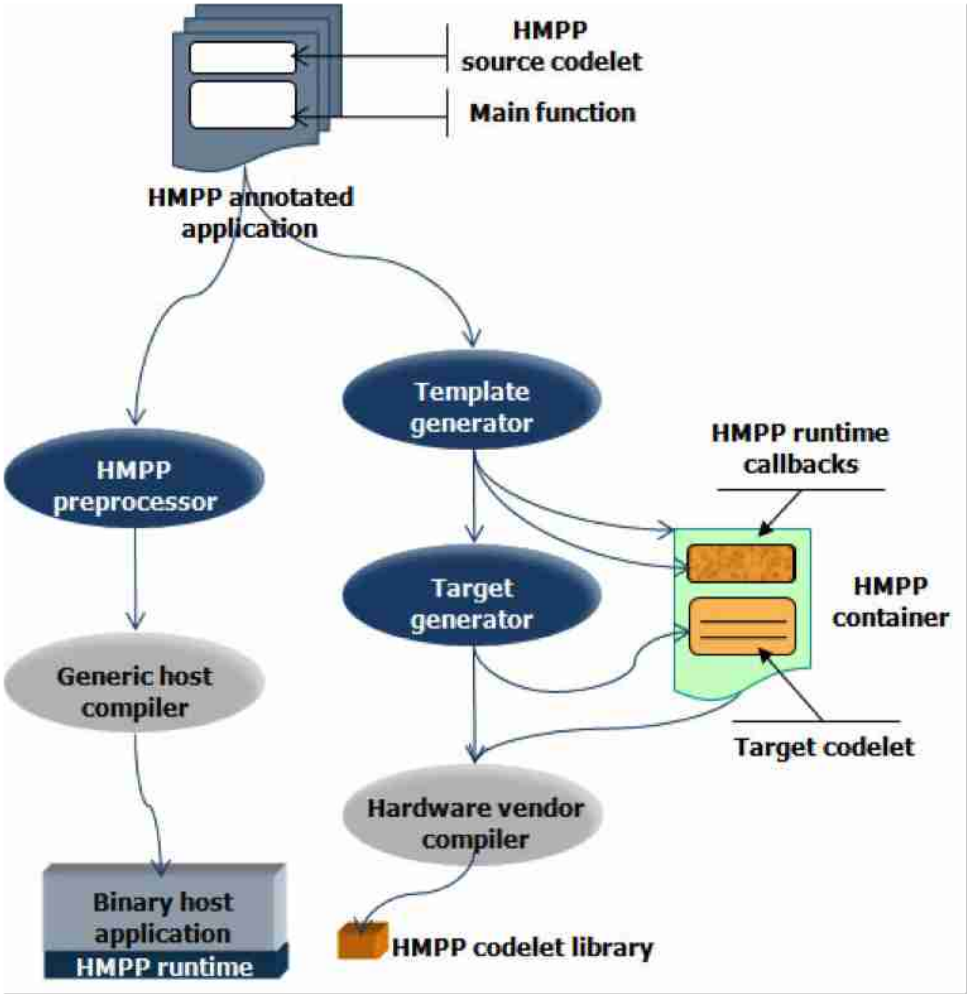
codelet / callsite directive set

Codelet Generation

Objectives

- Allow to transparently use HWA
  - From C or Fortran, Java to CUDA, Brook, ...
- Allow for code tuning at source code level
  - Directives based approach

Code Generation Flow



Codelet Generation

- C, Java or Fortran source code input
  - HWA oriented subset of the languages
- Set of directives to
  - Optimize target codelet generation
  - Express parallelism expression
- Make code tuning easier
- Generated code can also be tuned

Loop Parallelization

- Force or prevent the parallelization of loops
- Help defining kernels in a codelet

```
#pragma hmppcg parallel
for (i=0; i < n; i++) {
#pragma hmppcg noParallel
    for (j=0; j < n; j++) {
          D[i][j] = A[i][j] * E[3][j];
     }
}
```

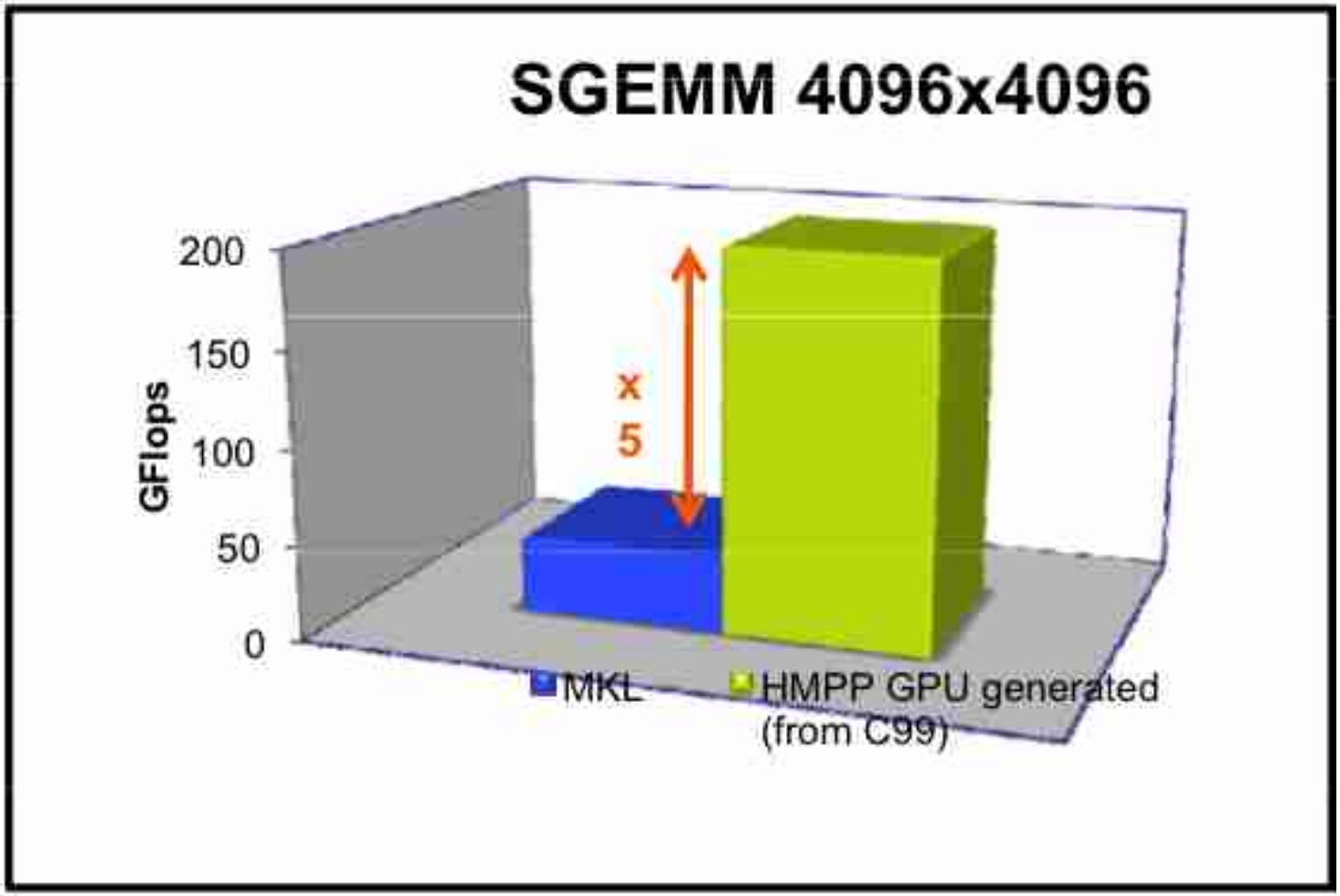
Input C Code Example 1

```
typedef struct{    float r, i;} Complex;
#pragma hmpp convolution2d codelet, args[data; opx].io=in, args[convr].io=out, target=CUDA
void convolution2d( Complex *data, int nx, int ny,  Complex *opx,
                    int oplx, int oply, Complex *convr) {
int hoplx = (oplx+1)/2;  int hoply = (oply+1)/2;
int iy, ix;
#pragma hmppcg parallel
for (iy = 0; iy < ny; iy++) {
#pragma hmppcg parallel
  for (ix = 0; ix < nx; ix++) {
    float dumr =0.0, dumi = 0.0;      int ky;
    for(ky = -(oply - hoply - 1); ky <= hoply; ky++) {
      int kx;
      for(kx = -(oplx - hoplx - 1); kx <= hoplx; kx++){
        int dx = min( max(ix+kx, 0), (nx - 1) );
        int dy = min( max(iy+ky, 0), (ny - 1) );
        dumr += data[dy * nx + dx].r * opx[(hoply - ky) * oplx + (hoplx - kx)].r;
        dumr -= data[dy * nx + dx].i * opx[(hoply - ky) * oplx + (hoplx - kx)].i;
        dumi += data[dy * nx + dx].r * opx[(hoply - ky) * oplx + (hoplx - kx)].i;
        dumi += data[dy * nx + dx].i * opx[(hoply - ky) * oplx + (hoplx - kx)].r;
       }
     }
     convr[iy*nx+ix].r = dumr; convr[iy*nx+ix].i = dumi;
   }
  }
}
```

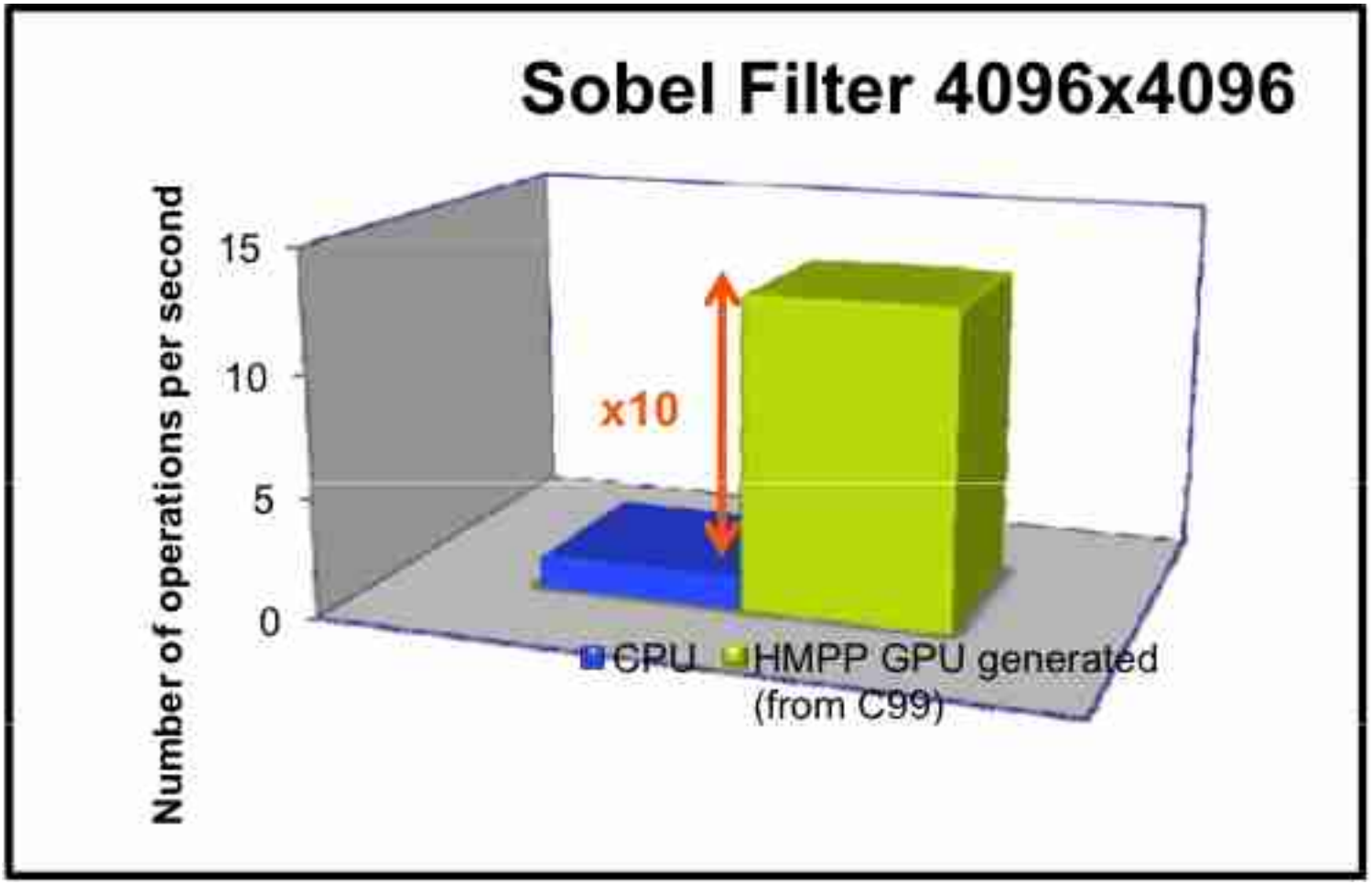
Input Fortran Code Example 2

```
!$HMPP sgemm3 codelet, target=CUDA, args[vout].io=inout
SUBROUTINE sgemm(m,n,k2,alpha,vin1,vin2,beta,vout)
INTEGER, INTENT(IN)    :: m,n,k2
REAL,   INTENT(IN)    :: alpha,beta
REAL,    INTENT(IN)    :: vin1(n,n), vin2(n,n)
REAL,    INTENT(INOUT) :: vout(n,n)
REAL     :: prod
INTEGER  :: i,j,k
!$HMPPCG unroll(8), jam(2), noremainder
!$HMPPCG parallel
DO j=1,n
    !$HMPPCG unroll(8), splitted, noremainder
    !$HMPPCG parallel
    DO i=1,n
        prod = 0.0
        DO k=1,n
          prod = prod + vin1(i,k) * vin2(k,j)
        ENDDO
        vout(i,j) = alpha * prod + beta * vout(i,j) ;
     END DO
END DO
END SUBROUTINE sgemm
```

MxM Performance



Performance Examples

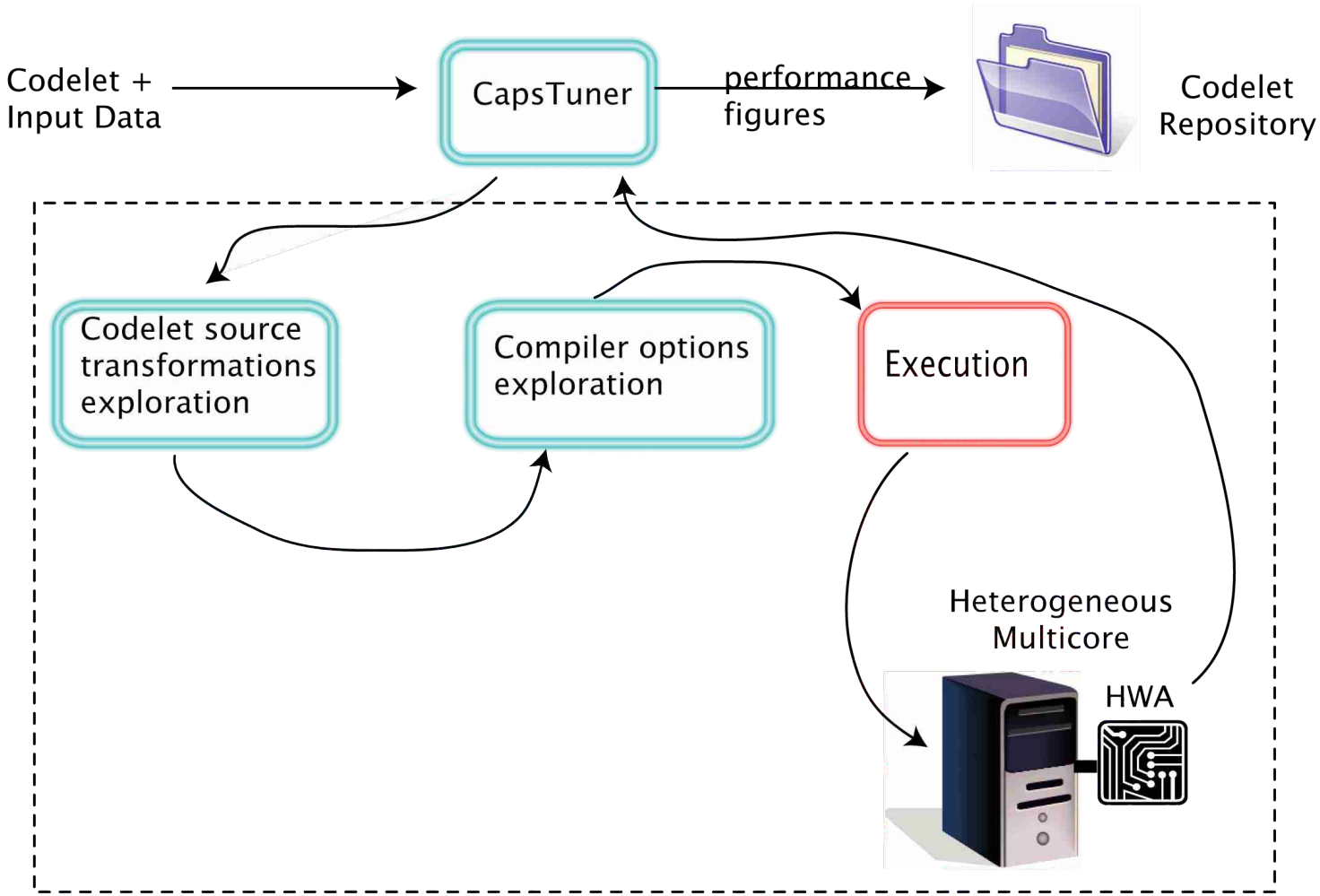


Codelet Tuning

Codelet Tuning (1)

- Based on CAPSTuner Technology
  - Platform independent
- Iterative compilation technique to explore the optimization space
  - Explore source code transformations
    - Via a set of code transformation directives such as unroll-and-jam
  - Explore compiler options
  - Store the performance data in a well defined repository

Code Tuning (2)



Other presentations

Optimisation d'applications
-
Composition de transformations de programme: modèle et outils

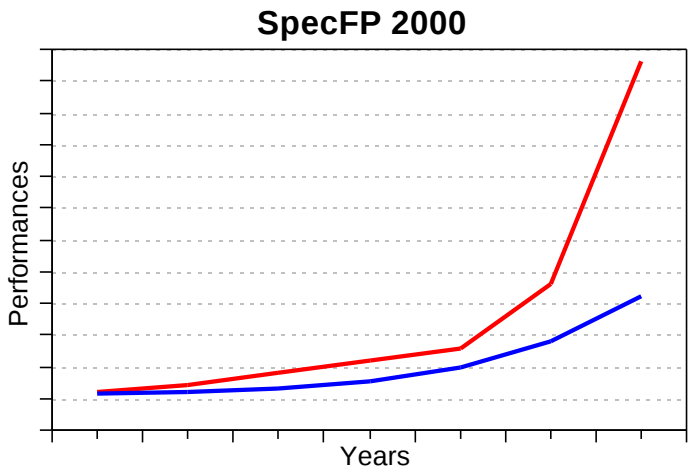
par Sylvain Girbal

co-directeur: Albert Cohen
co-directeur: Olivier Temam

encadrant au CEA: Jacques Raguideau

Context

- Increasing gap between peak performance and measured running time
- **Inefficient resource usage**



- Architecture
  - ↗ Speculation & complexity
    - Memory hierarchy
    - Branch predictors
    - Dynamic scheduling
    - Speculative Loads
    - Value prediction
  - ↗ Resources
    - Functional unit number
    - New units
- Compilation
  - Static cost models
    - Inefficient for speculative and dynamic mechanisms.
  - Loop nest transformations
    - Each defined in its own formalism
    - A few are implemented in compilers

Existing Optimization Frameworks

Optimization Tools:

- Mainly for parallelization of sequential codes
- Drawbacks
  - Black box behavior
  - Restricted Applicability

- Syntax based optimizers
  - Polaris
  - SUIF
  - ParaScope
  - Sage++

- Syntactic Representations
  - Growing complexity
  - Pattern matching rules
  - Phase behavior of compilers

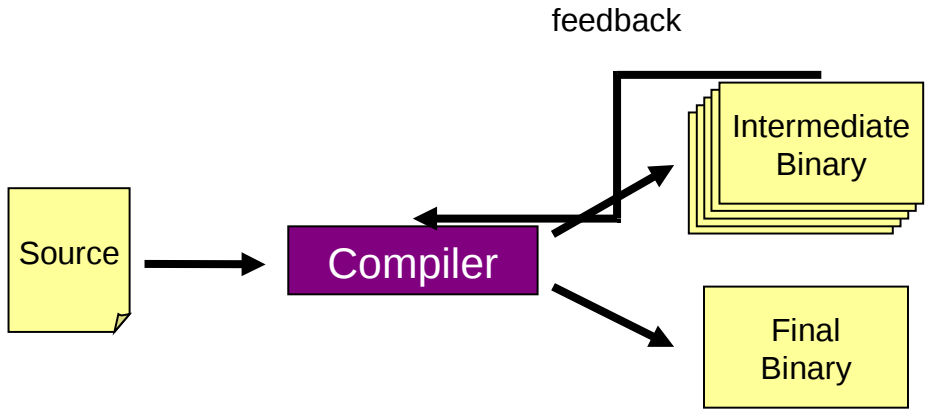
- Polyhedron based optimizers
  - Petit (Omega)
  - MMAlpha
  - PICO
  - PIPS

- Polyhedral Representation
  - Restricted applicability
  - Applicable to kernels only
  - Only implement a few transformations

Iterative Compilation

[O'Boyle]

- Embedded processors
  - Increased Compilation Time
  - Better optimizations
  - Dedicated architectures
  - Application with long lifespan



- General-Purpose processors
  - Searching best transformation parameters
  - Cope well with architectural changes
  - Efficient with sampling techniques

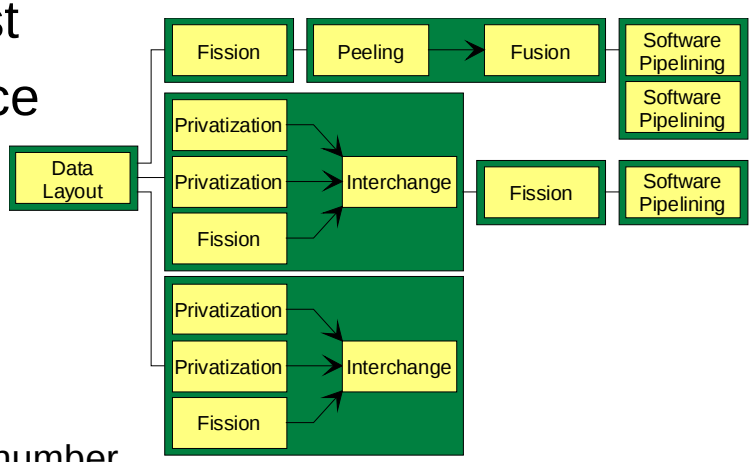
  - Only a few transformations are considered
  - Decisions mainly based on execution time
  - Restricted search space

Manual Optimizations

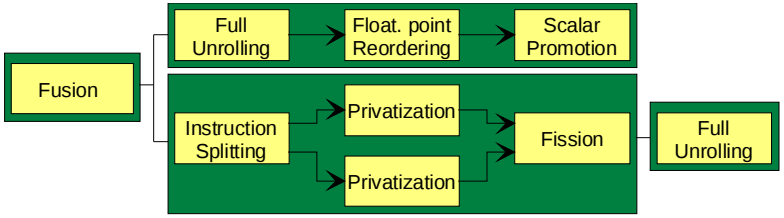
- Long sequences of transformations
- Transformations often hits the same loop nest
- Some “enabling” phases degrade performance

	peak	manual
swim	1,00	1,61
galgel	1,04	1,39
applu	1,47	2,18
apsi	1,07	1,23

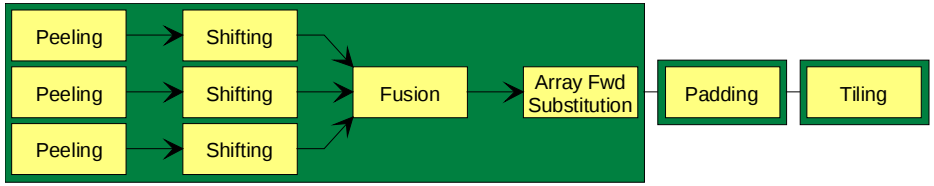
- Mainly regular loop nests
  - High number of loop nests
  - Low number of conditionals
- Performance loop nests
  - High variance in instruction number
  - Small loop depth



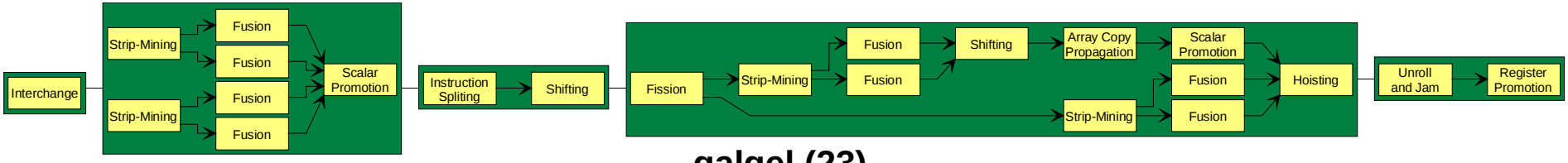
apsi (16)



applu (9)



swim (10)



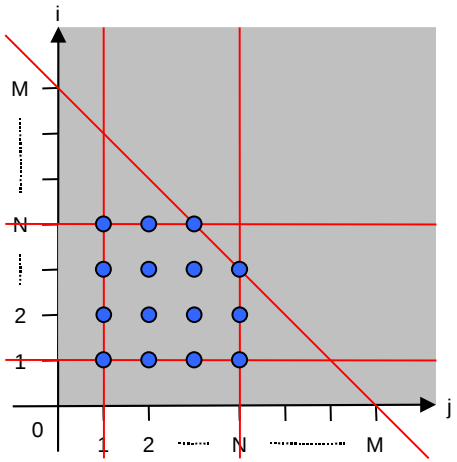
galgel (23)

Outline

- Code optimization context
- Unified Formalism
- Example of code optimization
- Compilation Process
- Conclusion

Introducing Polyhedra

- Dependency & Dataflow analysis
  - Fine grain (often exact) analysis enhance transformation opportunities
  - Based on $\mathbb{Z}$ -polyhedra



- Polyhedron = System of affine inequalities
  - { i≥1 ; i≤N ; j≥1 ; j≤N ; i+j≤M }

- Matrix representation

$$\begin{bmatrix} 1 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & -1 & 0 & 1 & 0 \\ -1 & -1 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} i \\ j \\ M \\ N \\ 1 \end{bmatrix} \geq 0$$

$$\begin{bmatrix} i-1 \\ N-i \\ j-1 \\ N-j \\ M-i-j \end{bmatrix} \geq 0$$

Defining Execution Order

[Feautrier]

- Statements are executed more than once
- Scheduling orders statement instances
  - < Statement , Iteration vector>
  - <S1, [3]> <S3, [8,9]>

```
for(int i=0;i<M;i++)
  S1
  S2
for(int i=0;i<M;i++)
  for(int j=0;j<N;j++)
    S3
```

All instances of statement S2 occur before any instance of statement S3
The instance <S1,i=7> occurs after instance <S2,i=6>

- Associate a timestamp to each instance
  - Scheduling function θ

- Mono-dimensional scheduling function
  - θ(S1,[i])=2*i θ(S2,[i])=2*i+1 θ(S3,[i,j])=2*M + i*N+j

Fusion ↳ θ(S1,[i])=(N+2)*i ↳ θ(S2,[i])=(N+2)*i+1 **Interchange** ↳ θ(S3,[i,j])=2*M+i*(N+2)+j

- Multi-dimensional scheduling function
  - Dimension d = depth(S)
    - θ(S1,[i])= [2*i] θ(S2,[i])= [2*i+1] θ(S3,[i,j])=[2*M+i,j]

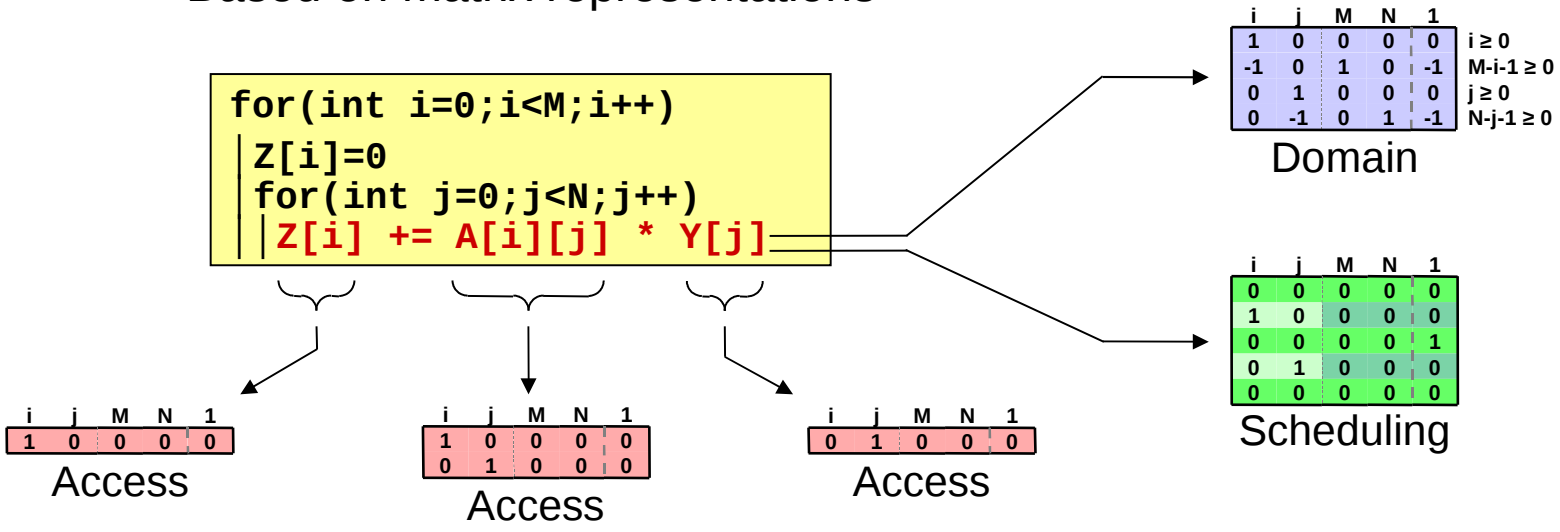
Fusion ↳ θ(S1,[i])=[3*i] ↳ θ(S2,[i])=[3*i+1] **Interchange** ↳ θ(S3,[i,j])=[2*M+i,j]

  - Dimension 2*d+1
    - θ(S1,[i])= [0,i,0] θ(S2,[i])= [0,i,1] θ(S3,[i,j])= [1,i,0,j,0]

Fusion ↳ θ(S1,[i])=[0,i,0] ↳ θ(S2,[i])=[0,i,1] **Interchange** ↳ θ(S3,[i,j])=[1,i,0,j,0]

Unified Formalism

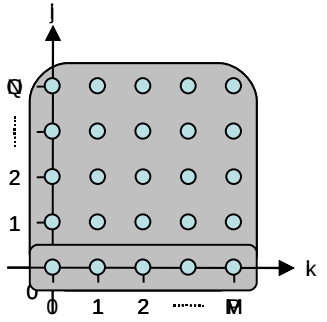
- Unified formalism for code transformations
  - Storing statement-wise information
  - Based on matrix representations



  - Code transformations = composition of matrix operations
  - Ease the composition of transformations

Statement Control

```
     for(int i=0;i<M;i++)
S1   |Z[i] = 0
     |for(int j=0;j<N;j++)
S2   ||Z[i] += (A[i][j] + B[j][i])*X[j]
     for(int k=0;k<P;k++)
     |for(int l=0;l<Q;l++)
S3   ||Z[k] += A[k][l] * Y[l]
```



$\Lambda^{S1} = \{ 0 \le i < M \}$

$\Lambda^{S1} = \{ i \ge 0 ; M-i-1 \ge 0 \}$

i	M	N	P	Q	1	
1	0	0	0	0	0	$i \ge 0$
-1	1	0	0	0	-1	$M-i-1 \ge 0$

$\Lambda^{S2} = \{ 0 \le i < M ; 0 \le j < N \}$

$\Lambda^{S2} = \{ i \ge 0 ; M-i-1 \ge 0 ; j \ge 0 ; N-i-1 \ge 0 \}$

i	j	M	N	P	Q	1	
1	0	0	0	0	0	0	$i \ge 0$
-1	0	1	0	0	0	-1	$M-i-1 \ge 0$
0	1	0	0	0	0	0	$j \ge 0$
0	-1	0	1	0	0	-1	$N-j-1 \ge 0$

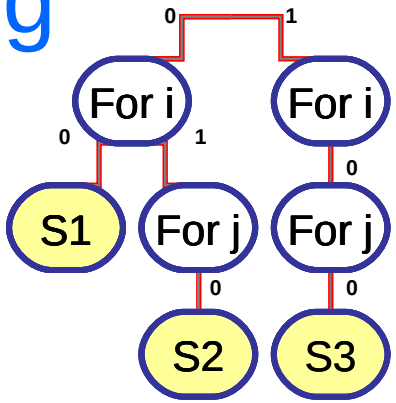
$\Lambda^{S3} = \{ 0 \le k < P ; 0 \le l < Q \}$

$\Lambda^{S3} = \{ k \ge 0 ; P-k-1 \ge 0 ; l \ge 0 ; Q-l-1 \ge 0 \}$

k	l	M	N	P	Q	1	
1	0	0	0	0	0	0	$k \ge 0$
-1	0	0	0	1	0	-1	$P-k-1 \ge 0$
0	1	0	0	0	0	0	$l \ge 0$
0	-1	0	0	0	1	-1	$Q-l-1 \ge 0$

Statement Scheduling

```
     for(int i=0;i<M;i++)
S1   |Z[i] = 0
     |for(int j=0;j<N;j++)
S2   ||Z[i] += (A[i][j] + B[j][i])*X[j]
     for(int i=0;i<P;i++)
     |for(int j=0;j<Q;j++)
S3   ||Z[j] += A[j][i] * Y[j]
```



$\Theta^{S1} = \begin{pmatrix} 0 \\ i \\ 0 \end{pmatrix}$

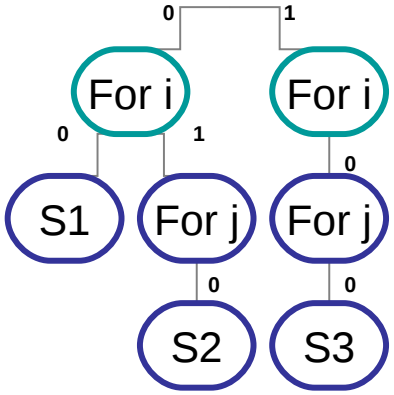
i	M	N	P	Q	1
0	0	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0

$\Theta^{S2} = \begin{pmatrix} 0 \\ i \\ 1 \\ j \\ 0 \end{pmatrix}$

i	j	M	N	P	Q	1
0	0	0	0	0	0	0
1	0	0	0	0	0	0
0	0	0	0	0	0	1
0	1	0	0	0	0	0
0	0	0	0	0	0	0

$\Theta^{S3} = \begin{pmatrix} 1 \\ i \\ 0 \\ j \\ 0 \end{pmatrix}$

i	j	M	N	P	Q	1
0	0	0	0	0	0	1
1	0	0	0	0	0	0
0	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0



Scheduling : Separation

$\Theta^{S3} =$

i	j	M	N	P	Q	1
0	0	0	0	0	0	1
1	0	0	0	0	0	0
0	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

i	j	M	N	P	Q	1
0	0	0	0	0	0	1
0	0	0	0	0	0	0
0	0	0	0	0	0	0

i	j	M	N	P	Q	1
1	0	0	0	0	0	0
0	1	0	0	0	0	0

$\beta^{S3} =$

1
0
0

Scheduling vector

- Fusion
- Fission
- Code motion

$A^{S3} =$

i	j
1	0
0	1

Iteration scheduling matrix

- Interchange
- Skewing
- Reversal

$\Gamma^{S3} =$

M	N	P	Q	1
0	0	0	0	0
0	0	0	0	0

Parameter scheduling matrix

- Shifting

Outline

- Code optimization context
- Unified Formalism
- Example of code optimization
- Compilation Process
- Conclusion

Code Optimization : Example

```
   for(i=0;i<M;i++)
S1 |Z[i] = 0
   |for(j=0;j<M;j++)
S2 ||Z[i] += (A[i][j] + B[j][i])*X[j]
   for(i=0;i<M;i++)
   |for(j=0;j<M;j++)
S3 ||Z[i] += A[i][j] * Y[j]
```

```
External loop fusion
Internal loop fusion
Fission of Z initialization
Internal strip-mine
External strip-mine
Center loop interchange
```

Fusion

```
   for(i=0;i<M;i++)
S1 |Z[i] = 0
   |for(j=0;j<M;j++)
S2 ||Z[i] += (A[i][j] + B[j][i])*X[j]
   |for(j=0;j<M;j++)
S3 ||Z[i] += A[i][j] * Y[j]
```

Fusion

```
   for(i=0;i<M;i++)
S1 |Z[i] = 0
   |for(j=0;j<M;j++)
S2 ||Z[i] += (A[i][j] + B[j][i])*X[j]
S3 ||Z[i] += A[i][j] * Y[j]
```

Fission

```
   for(i=0;i<M;i++)
S1 |Z[i] = 0
   for(i=0;i<M;i++)
   |for(j=0;j<M;j++)
S2 ||Z[i] += (A[i][j] + B[j][i])*X[j]
S3 ||Z[i] += A[i][j] * Y[j]
```

Tiling

```
   for(i=0;i<M;i++)
S1 |Z[i] = 0
   for(ii=0;ii<M/32;ii++)
   |for(jj=0;jj<M/32;jj++)
   ||for(i=32*ii;i<min(M,ii*32+32);i++)
   |||for(j=32*jj;j<min(M,jj*32+32);j++)
S2 ||||Z[i] += (A[i][j] + B[j][i])*X[j]
S3 ||||Z[i] += A[i][j] * Y[j]
```

Code Optimization : Example

```
    for(i=0;i<M;i++)
S1  |Z[i] = 0
    |for(j=0;j<M;j++)
S2  ||Z[i] += (A[i][j] + B[j][i])*X[j]
    for(i=0;i<M;i++)
    |for(j=0;j<M;j++)
S3  ||Z[i] += A[i][j] * Y[j]
```

External loop fusion
Internal loop fusion
Fission of Z initialization
Internal strip-mine
External strip-mine
Center loop interchange

$A^{S1} =$ | i |
|---|
| 1 |

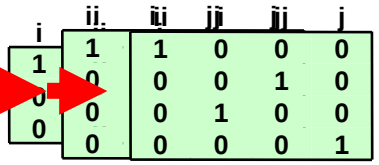
$\beta^{S1} =$ | 0 |
|---|
| 0 |

$\Lambda^{S1} =$

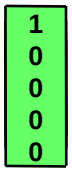
i	M	1
1	0	0
-1	1	-1

$A^{S2} =$

i	j
1	0
0	1



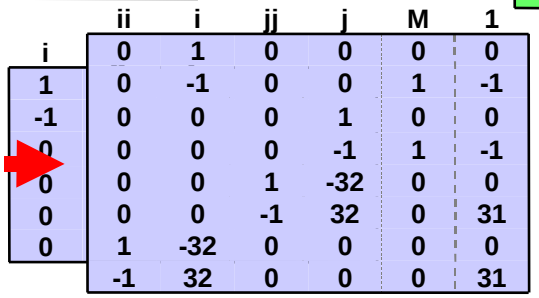
$\beta^{S2} =$ | 0 |
|---|
| 1 |
| 0 |



1
0
0
0
0

$\Lambda^{S2} =$

i	j	M	1
1	0	0	0
-1	0	1	-1
0	1	0	0
0	-1	1	-1

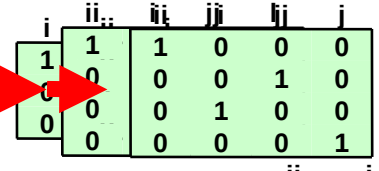


i
1
-1
0
0
0
0

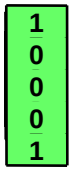
ii	i	jj	j	M	1
0	1	0	0	0	0
0	-1	0	0	1	-1
0	0	0	1	0	0
0	0	0	-1	1	-1
0	0	1	-32	0	0
0	0	-1	32	0	31
1	-32	0	0	0	0
-1	32	0	0	0	31

$A^{S3} =$

i	j
1	0
0	1



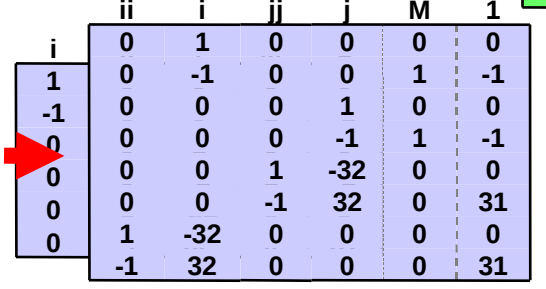
$\beta^{S3} =$ | 1 |
|---|
| 0 |
| 0 |



1
0
0
0
1

$\Lambda^{S3} =$

i	j	M	1
1	0	0	0
-1	0	1	-1
0	1	0	0
0	-1	1	-1



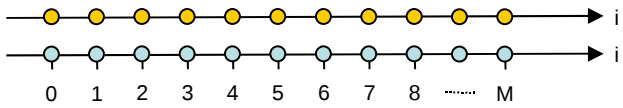
i
1
-1
0
0
0
0

ii	i	jj	j	M	1
0	1	0	0	0	0
0	-1	0	0	1	-1
0	0	0	1	0	0
0	0	0	-1	1	-1
0	0	1	-32	0	0
0	0	-1	32	0	31
1	-32	0	0	0	0
-1	32	0	0	0	31

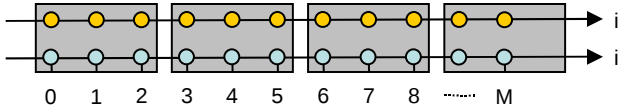
```
    for(i=0;i<M;i++)
S1  |Z[i] = 0
    for(ii=0;ii<M/32;ii++)
    |for(jj=0;jj<M/32;jj++)
    ||for(i=32*ii;i<min(M,ii*32+32);i++)
    |||for(j=32*jj;j<min(M,jj*32+32);j++)
S2  ||| Z[i] += (A[i][j] + B[j][i])*X[j]
S3  ||| Z[i] += A[i][j] * Y[j]
```

Composition Issue: Strip Mining case

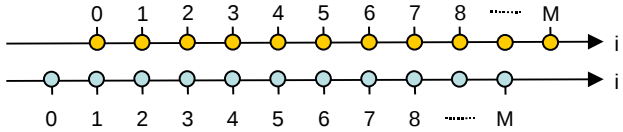
- Details on Strip-mining
  - Strip-mine

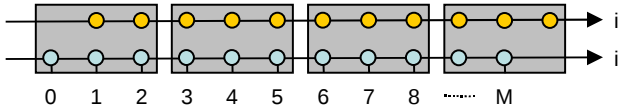



```
for(i=0;i<=M;i++)
  S1(i)
  S2(i)
```

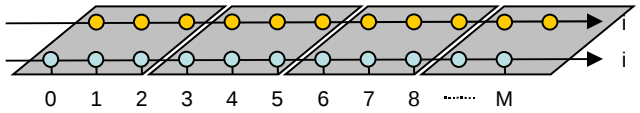


  - Shifting + Strip-mine





    - Classical Strip-mining
      - Parallel to domain iterators

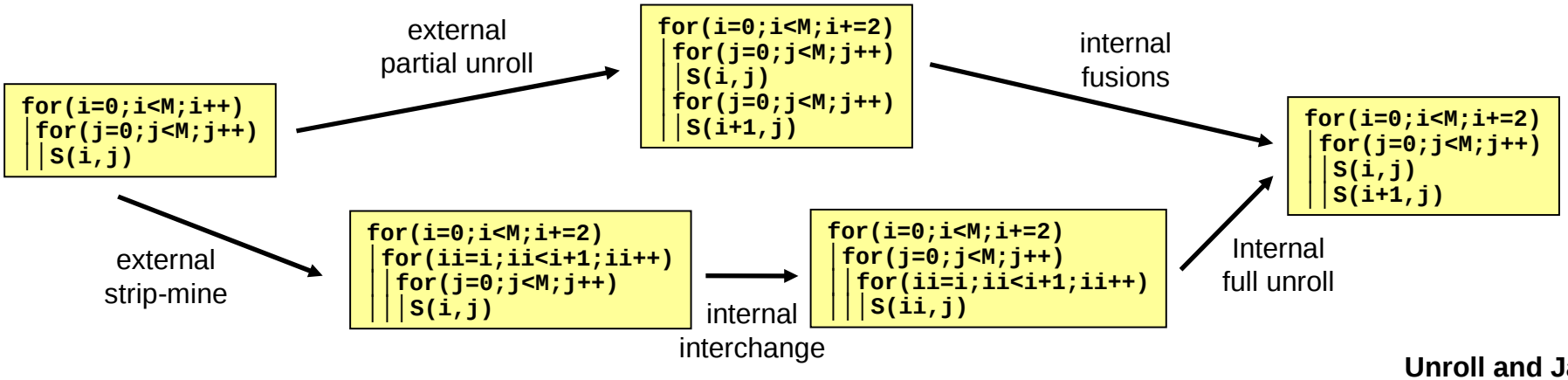


    - Time Strip-mining
      - Parallel to traversal order
      - Usually preferred

Composition & Confluence

- Commutation
  - Transformation targeting different components commute
    - Scheduling transformation
      - Transformation changing sequentiality (β-Transformation)
      - Transformation changing iteration ordering (A-Transformation,Γ-Transformation)
    - Domain transformation (Λ-Transformation)
    - Access transformation
  - β-Transformations commutes with other β-Transformations
  - Dimension transformations do not commute

- Confluence

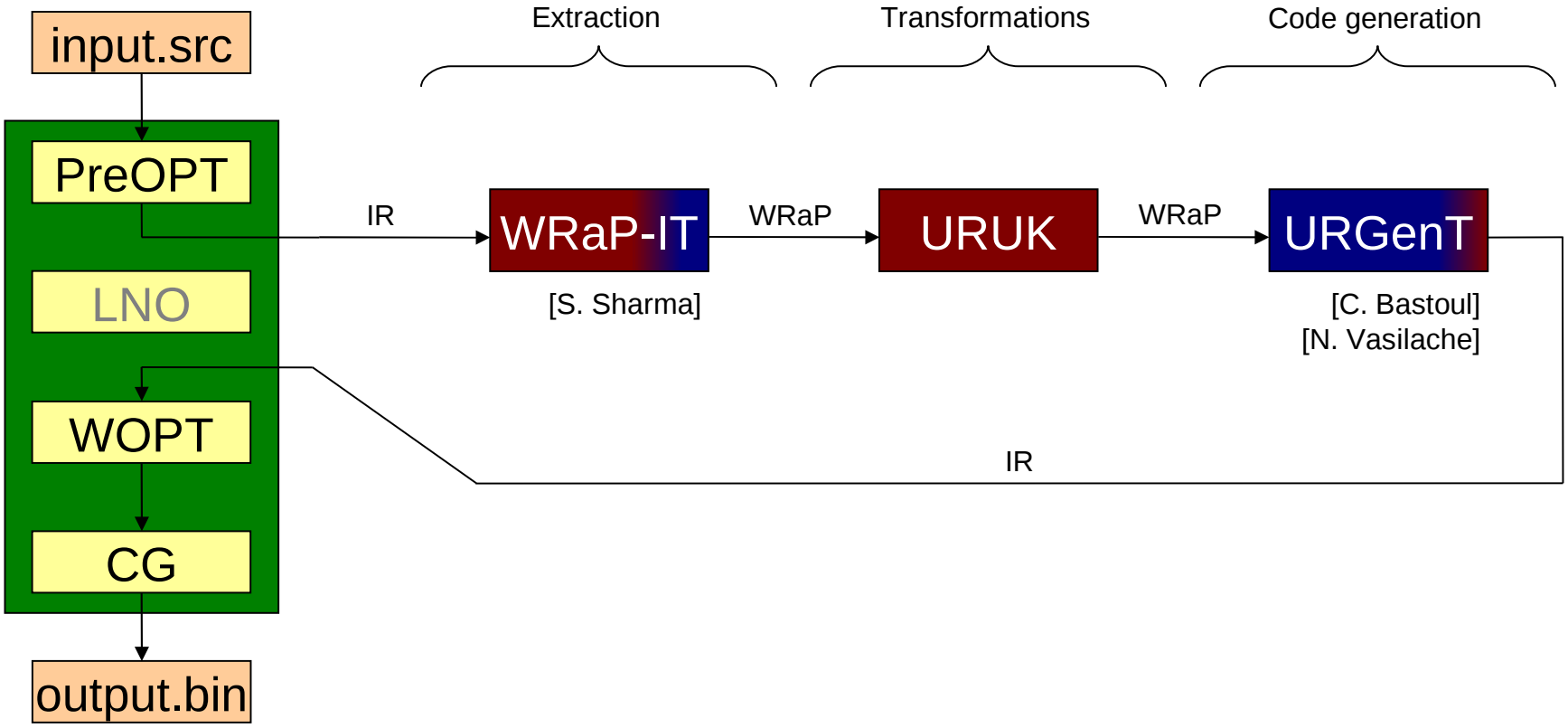


Unroll and Jam

Outline

- Code optimization context
- Unified Formalism
- Example of code optimization
- Compilation Process
- Conclusion

Compilation Process



Defining Transformations

- With a Script language to easily add some new transformations.
- As composition of previously defined transformation.
- Using C++ to keep syntax close to the formalism.

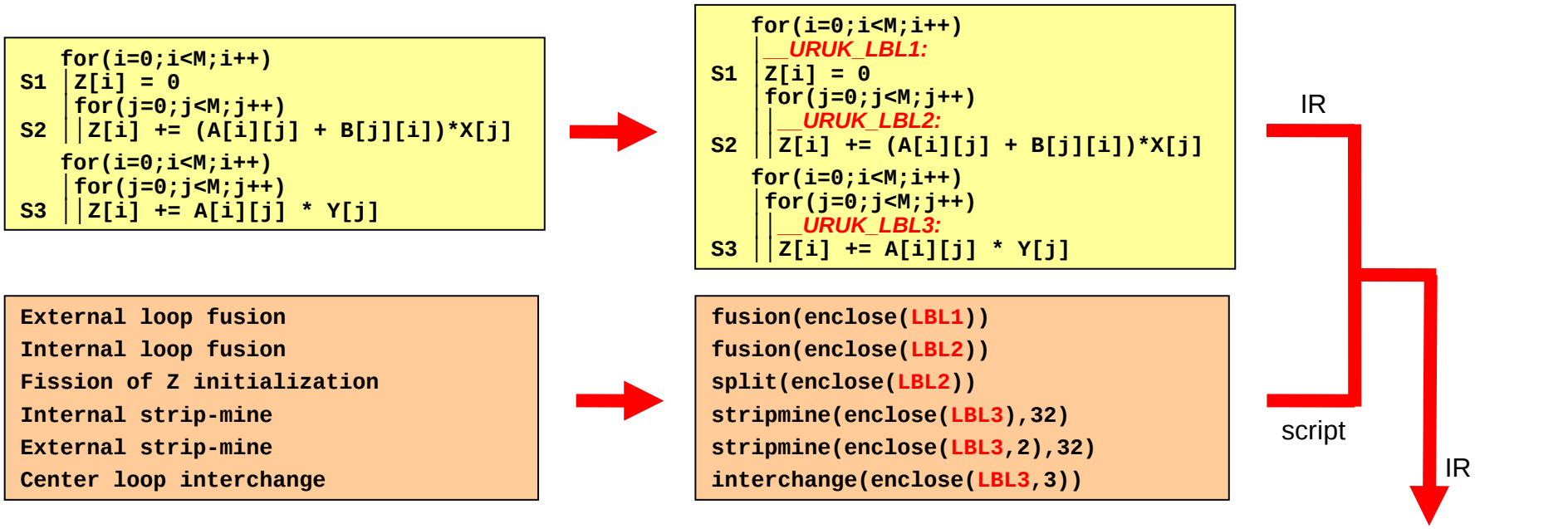
$$\text{move}(P,Q,o)$$
$$\quad d \leftarrow \dim(P)$$
$$\quad \forall S \in \text{SCoP}$$
$$\quad\quad P \subseteq \beta^S \wedge Q \subseteq \beta^S \Rightarrow \beta^S_d \leftarrow \beta^S_d + o$$
$$\quad\quad P \subseteq \beta^S \wedge Q << \beta^S \Rightarrow \beta^S_d \leftarrow \beta^S_d + o$$

```
%transformation move

%param BetaPrefix P
%param BetaPrefix Q
%param Integer o

%code
{ d=P.dim();
  foreach WrapStatement S in SCoP
  {  if ((P<=S.Beta) && (Q<=S.Beta))  S.Beta(d)+=o;
     if ((P<=S.Beta) && (Q<<S.Beta))  S.Beta(d)+=o;
  }
}
```

Applying Transformations



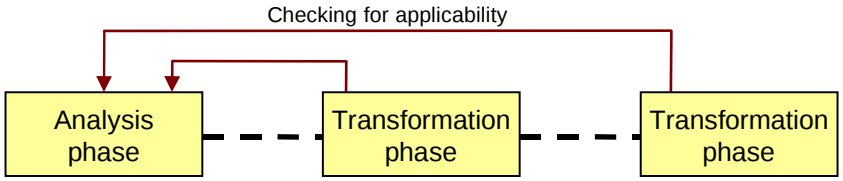
- Framework inputs:
  - Source code
    - Decorated with labels
    - Using compiler internal representation
  - Script
    - Describing transformation to apply
    - Using source labels
- Framework output
  - Transformed IR

```
   for(i=0;i<M;i++)
S1 |Z[i] = 0
   for(ii=0;ii<M/32;ii++)
   |for(jj=0;jj<M/32;jj++)
   ||for(i=32*ii;i<min(M,ii*32+32);i++)
   |||for(j=32*jj;j<min(M,jj*32+32);j++)
S2 |||| Z[i] += (A[i][j] + B[j][i])*X[j]
S3 |||| Z[i] += A[i][j] * Y[j]
```

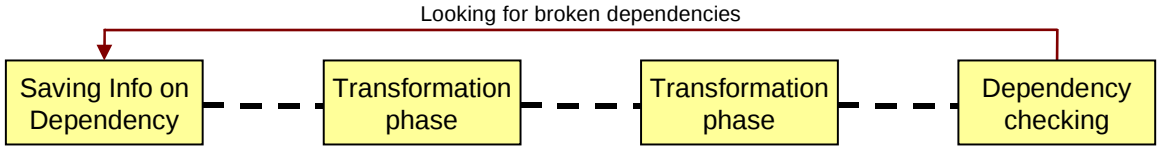
Tools using the Formalism

- PolyDeps : Dependency checker

Traditional:



PolyDeps:



- URGenT : Code Generator
  - Based on CLooG
  - Taking advantage of formalism invariants
  - Exponential reduction in the number of polyhedral computation
  - Reduce memory trace

Conclusion & Future Works

- Contributions
  - Program abstraction class : SCoPs
    - Good coverage for non pointer intensive codes
  - Formalism for both program and program transformations
    - Component separation
    - Eases composition of program transformations
  - Implementation of the compilation framework
    - Definitions close to the formalism, composition oriented
    - Usable by non-expert users
  - swim SpecFP 2000 benchmark
    - More than 30% speedup compared to best compilers.

Conclusion & Future Works

- Ongoing & Future Works
  - Optimize more spec benchmarks **automatically**
  - Searching for transformation opportunity
    - Opportunities as transformations (code motion)
  - Engineering
    - WRaP-IT enhancements (modulo) better SCoP coverage
    - Array generation How to provide array memory mapping information
    - A language for URUK Scripts Polyhedral meta-programming
  - Extending the notion of labels
    - Miss a way to target range of source code
    - Toward “instruction instances”
  - Integration in an iterative compilation framework

Denis Barthou Slides

ITEA 2

INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

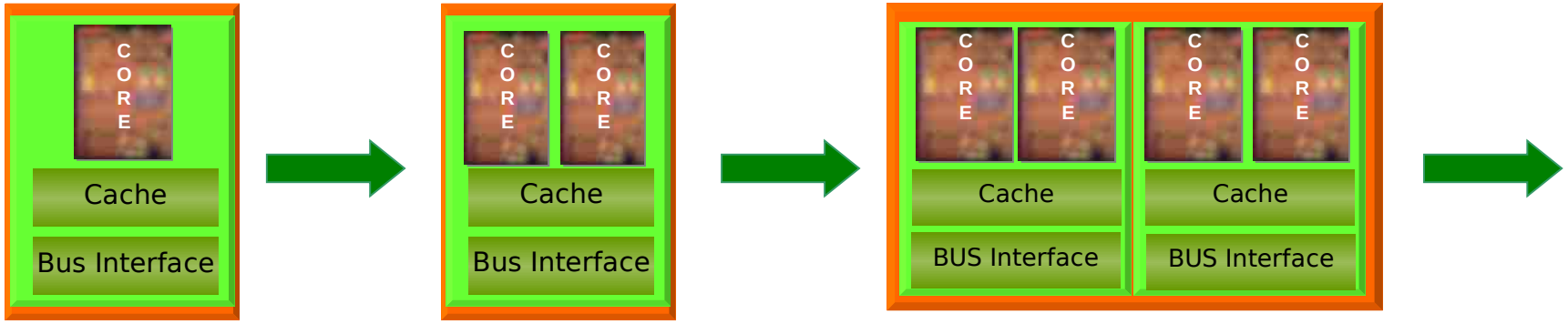
Optimizing Applications for Multicore

in the ParMA Project

Par*allel Programming for* ***M****ulti-core* ***A****rchitectures*

www.parma-itea2.org

Software and the Concurrency Revolution by Herb Sutter and James Larus, Microsoft



Single-core Chip 2005 **Dual-core Chip 2006** **Quad-core Chip 2007**

- **Enhanced computing power**
  - **Not from using one faster core (processor unit)**
  - **But from using several cores**
- **Problem: This requires transition from sequential to parallel programming**

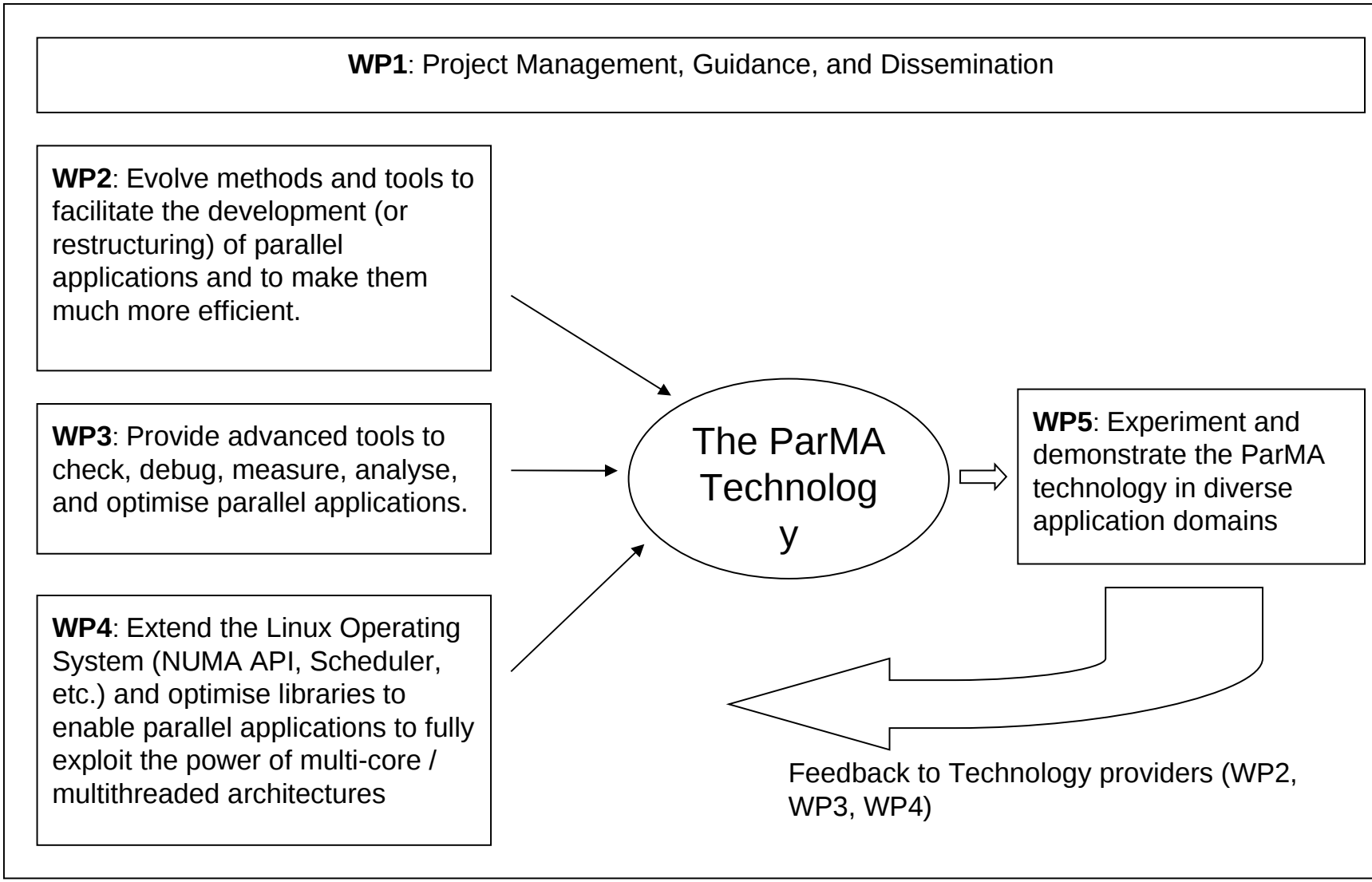
Because a lot of domains depends heavily on the use of efficient technical and scientific computing systems, parallel computing is becoming more and more important for the European society and industry,

but programmers aren't well equipped
to parallelize and optimize code on multicore.

- There are various types of massively and multilevel parallel architectures
- There is no silver bullet for efficient parallel programming

=> Several difficult problems must be tackled

- To meet new scientific and engineering challenges, we mainly need:
  - faster, more accurate computational mathematics
  - new parallel programming models,
  - enriched directives to express parallelism,
  - methods and tools to detect and extract parallelism (thread extraction),
  - easy to use and scalable profiling, optimization, and memory management tools,
  - powerful correctness checker and debugger,
  - optimized libraries for multithread / multi-core,
  - enhanced thread management, job scheduling, ...


WP1: Project Management, Guidance, and Dissemination
WP2: Evolve methods and tools to facilitate the development (or restructuring) of parallel applications and to make them much more efficient.
WP3: Provide advanced tools to check, debug, measure, analyse, and optimise parallel applications.
WP4: Extend the Linux Operating System (NUMA API, Scheduler, etc.) and optimise libraries to enable parallel applications to fully exploit the power of multi-core / multithreaded architectures
The ParMA Technology
WP5: Experiment and demonstrate the ParMA technology in diverse application domains
Feedback to Technology providers (WP2, WP3, WP4)

- Evolve design and programming models for parallel applications
  - Multi-level parallel programming & execution model for MPSoC by **CEA-LIST / LLSP**
  - Hybrid parallel programming: convert OpenMP programs to (OpenMP + MPI) by **INT**
  - Application restructuring by **CAPS** and **UVSQ**
  - Adaptation of message passing APIs to MPSoC environments by **UAB**
  - Method & Tool to design interconnection Networks on Chip (NoC) by **UAB** & **Robotiker**
- Provide parallel programming tools that apply to any application domain and any parallel architecture, and integrate them
  - Correctness tool: **MARMOT** from **HLRS**, to support hybrid applications (OpenMP + MPI)
  - Tracing facilities: Open Trace Format; Peruse (MPI extension)
  - Debugger: **DDT** from **Allinea** to deal with large number of threads and OpenMP
  - Measurement & Analysis: **KOJAK** from **JSC**; **Vampir** (visualizer) from **TU Dresden**

- Push operating systems to support new multi-core architectures
  - Linux extensions by **Bull:**
    - Improve shared and cache memory management to minimize buffer copy and optimize bandwidth
    - Improving cache coherency for better latency
    - Monitor timer, etc.

- Demonstrate the technology in various application domains
  - Metal forming simulation with INDEED by **GNS**
  - Casting process simulation with MAGMASOFT by **Magma**
  - 3D-combustion simulation by RECOM-AIOLOS by **RECOM**
  - HPC benchmark by **Bull**
  - Avionics (Tracking algorithm; Controlled simulation of system) by **Dassault Aviation**
  - Software Defined Radio by **Indra**
  - Virtual reality for manufacturing processes: optimisation of Collision Detection) by **CEA-LIST / LSI**

- All tools and most applications have been installed on the common Bull HPC cluster
- Application developers have experimented with the various tools
- Test and Performance tools have been evolved and start being integrated (e.g. KOJAK and Vampir, Marmot and DDT)
- The code optimisation tool (MAQAO) is being adapted for Xeon
- Performance analysis has been carried out by all applications to measure the speed-up on a multicore architecture
- Restructure and parallelisation of applications has started
- HPC benchmarks are being used to experiment with diverse MPI libraries
- Requirements for the FPGA / MPSoC to be used for improving SDR have been defined and the NoC design tool is available

- Exploiting the power of multi-core is not straightforward
  - With the level of parallelism implemented in a classical HPC application, the speed-up is often below the ideal (e.g. 2.4 on 8 cores with OpenMP)
  - Experience with an hybrid (OpenMP + MPI) application show that only with a given combination of OpenMP threads and MPI processes, a parallel efficiency of 94% can be obtained

  ⇒ Detailed analysis using the performance tools is required

  ⇒ Parallel strategies must be improved

  ⇒ In some cases new algorithms must be developed

  Technical meeting are regularly convened to share about this

- **CEA-LIST** conducted a research to increase the performance of **Collision Detection** (a complex tree traversal problem due to the heterogeneity of individual tasks w.r.t. compute time) using BVH (Bounding Volume Hierarchy).
- They develop a task calibration scheme to get a set of homogeneous tasks matching the number of available cores that limits the synchronisation overhead.
- Among their findings:
  - one can optimize DFS (Depth First Search) by exploiting the parallelism discovered by first traversing the tree in a weight directed manner.
  - driving the tree exploration relying on geometrical metric succeeds in extracting parallelism but remains too sensitive to over-parallelization.
  - re-introducing temporal information allows to drastically limit the over parallelization and its induced overhead.

  => In an industrial test case carried out on a 8 core machine with an initial speed-up of 1.7, their extraction parallelism (mix of temporal coherency and geometrical properties) enabled to deliver a speed-up of 6.9.

- A compléter éventuellement avec:
  - Des résultats du WP2 (C. Aussages, W. Jalby)
  - Quelques exemples d’amélioration des outils

Objectives:

1) Identify real performance bottlenecks

2) Find how to transform application to remove them

Multi-scale optimization required

- Parallelism between nodes, cores, threads, and ILP
- Performance results in good combination of optimization at different levels
  - Scalability issue may be explained by sequential kernels using high memory bandwidth
  - Memory bandwidth usage depends on low level optimizations (prefetch, copies, ...)

Different tools for different approaches

- Collecting exection traces and profiling
  - Vampir, openTrace
- Profile-guided tuning tools
  - Marmot, Kojak, MAQAO
- Static Performance Evaluation
  - MAQAO

Building blocks for multicore optimization

Issues

- Which interactions
  - For identifying bottlenecks
  - For optimizations

Objective of the tool

- Identify performance bottleneck of thread code
- Propose performance tuning hints

Objective of the tool

- Identify performance bottleneck of one thread code
- Propose performance tuning hints

Identifying single thread performance issues

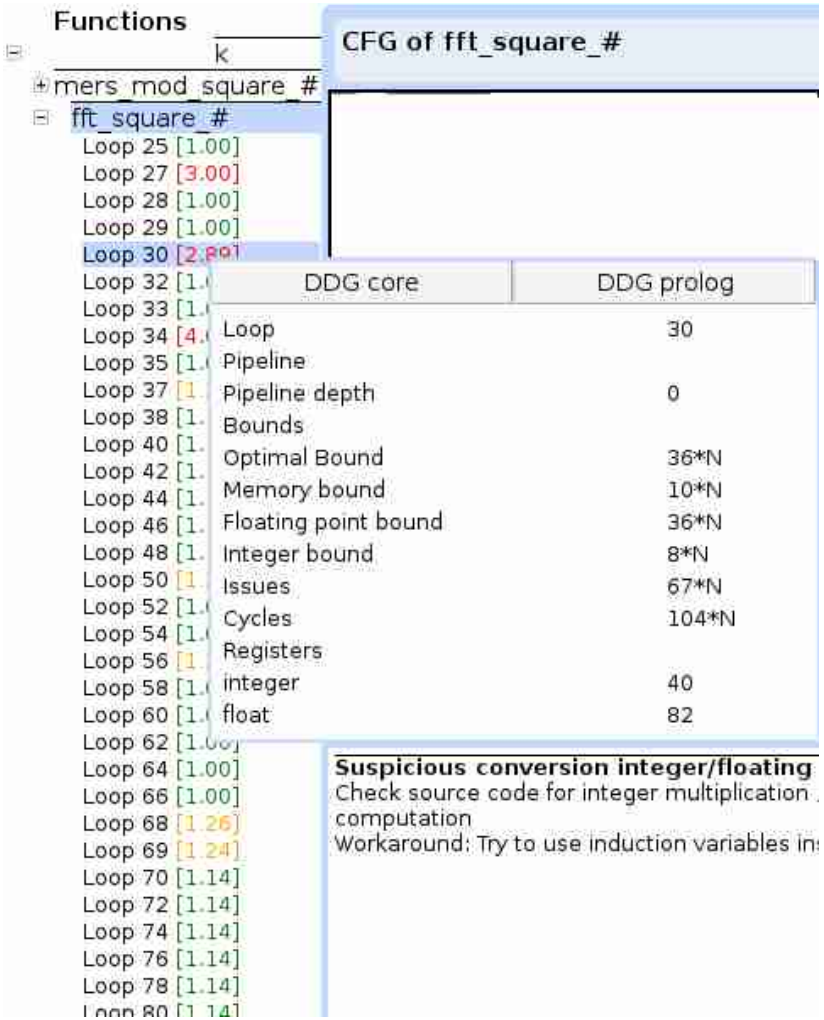
- Collect static information from assembly codes
  - Detect inefficient code patterns
  - Static performance evaluation for inner loops
- Collect value profiling information
  - Function parameters,
  - Prefetch values,
  - Loop trip count distribution...

Estimate potential performance gain

- For each inner loop
- For each function hotpath

Find opportunities for specialization

- Loop trip counts
- Function parameters

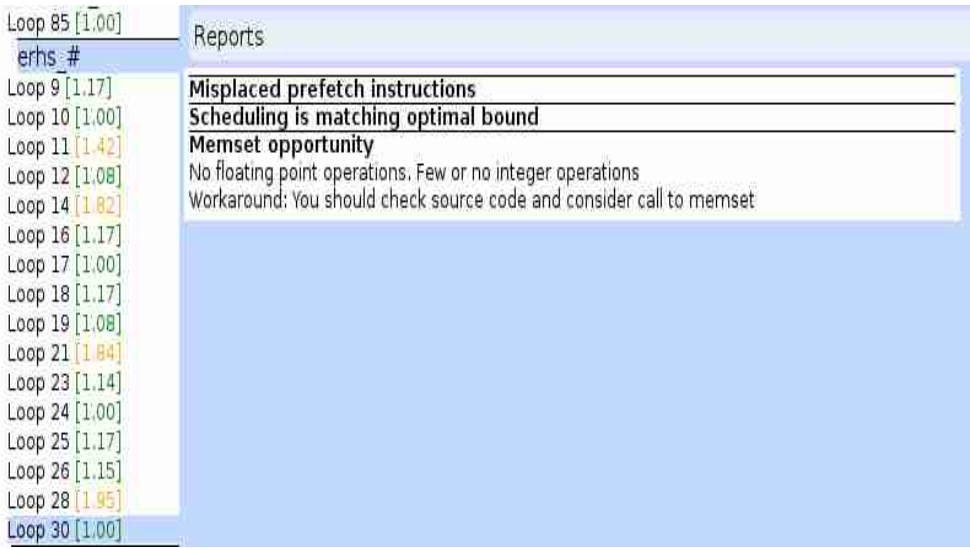


Performance tuning hints

- Source code modifications
- Compiler flags/pragmas
- Specialization opportunities

Limitation

- **No complex optimization (one loop)**



Exploring many optimizations

- Specify optimization space to explore with pragmas, for a given code fragment

  [Xlanguage, LCPC05, Donadio *et al*.]

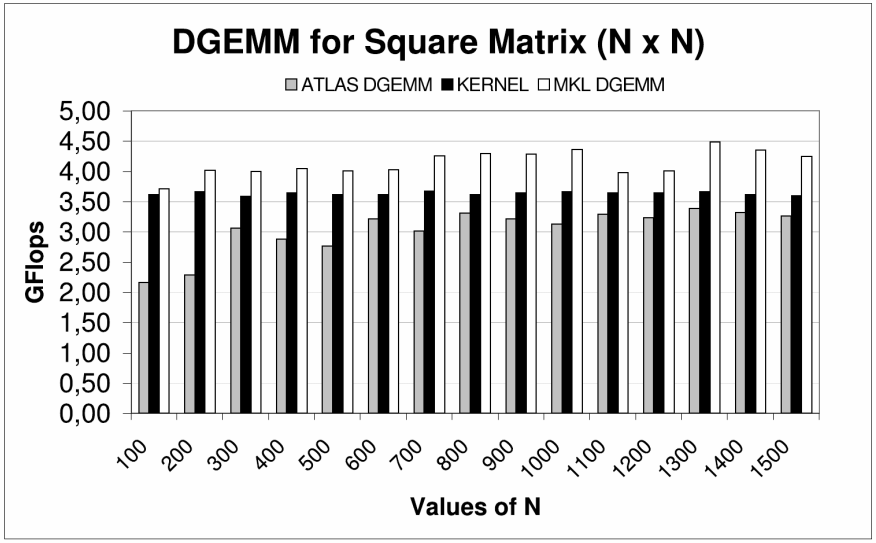
- Source to source transformations: unroll, interchange, tiling, ...
  - Can be user-defined, application dependent
  - Optimization space user-defined
- Generate multiple kernels versions

Combining with MAQAO

- Keep best performing kernels
  - Filtering out worst codes with MAQAO based on static performance evaluation
  - Benchmarking each kernel
    - Relevant if kernel is regular code => performance does not depend on input

Library generation

- For linear algebra, performance comparable to vendor-libraries (Intel MKL forIA64 and IA32)



[CGO07, Barthou *et al.*]

Application Performance Tuning

- Using hierarchical optimization and MAQAO tool to focus on bottleneck issues
- Molecular simulation code (GIBBS)
  - Speed up of x2.24 on Itanium2 platform
- Lattice QCD (ETMC code)
  - Large floatting point code (+1500 flop in hot loop)
  - Speed up of 40% for lattice 16^4

Towards a bottom up approach

- Previous approach generates multiple code versions to choose from
- For each version
  - Performance estimation
  - Memory footprint
- Possible to pick best version
  - For scalability
  - Satisfying thread affinity constraints
  - Addressing communication issues

Future work

- Integration of different tools (Maqao/Kojak)
- Global methodology for multicore performance tuning

More information:

http://www.parma-itea2.org

- Backup slides

I) An assortment of end-users

- Several industrial partners (including 3 German SMEs) working in different domains but all **with a strong need for high performance computing**:
  - **MAGMA** (D), to simulate complex casting processes
  - **RECOM** (D), to model 3D-combustion in industrial furnaces and boilers
  - **GNS** (D) for car crash analysis and forming simulation
  - **INDRA** (S) and **Dassault Aviation** (F) to develop a new generation of avionic embedded systems (for navigation, anti-collision, air traffic management, weather radar, etc.) and tactical communications systems
- An Applied Research Lab developing innovative solutions for major industrial players in the nuclear, automotive, aeronautical, …, medical fields needing very high performance in computation times:
  - **CEA-LIST / LSI** (F) will also contribute to requirement definition and then to experimentation of the ParMA technology by improving a physically realistic real time simulation of an automotive manufacturing process

II) A platform provider

- **Bull** (F), the French computer manufacturer, that builds HPC computers is developing an extended version of the Linux Operating System that will enable parallel applications to exploit fully the power of multi-core architectures on top of which parallel programming tools will be integrated to help customers develop, debug, and tune parallel applications.

III) Three High Performance Computing Centres

- **HLRS / Universität Stuttgart** (D) with a long experience in the development of parallel tools and libraries (e.g. MPI-correctness tool MARMOT and the meta-computing library PACX-MPI) in close cooperation with users of supercomputing resources.
- **ZIH / TU Dresden** (D) an established competence centre for parallel computing and software tools that supports users from the industry as well as research institutes with a focus on performance evaluation and optimization (performance analysis tools Vampir, VampirTrace, Open Trace Format / OTF).
- **JSC** (D) (Jülich Supercomputing Center) which runs the most powerful scientific computer centres in Europe, is carrying out research on several supercomputing topics, in particular on performance analysis tools for parallel programming, in particular with the KOJAK tool (**K**it for **O**bjective **J**udgement & **A**utomatic **K**nowledge-based detection of bottlenecks).

IV) Three SMEs offering services, or developing and marketing parallel programming tools

- **CAPS-Entreprise** (F) specialised in high performance and embedded computing, especially in restructuring of sequential programs: execution traces are analysed to find independent sequences that could be executed in parallel

- **GWT** (D), which conducts commissioned research for industry and public sector, is responsible for the development, marketing and support of the performance analysis framework Vampir + OTF.

- **Allinea** (UK) a leading supplier of tools for high performance computing, such as the Distributed Debugging Tool (DDT) and the Optimization and Profiling Tool (OPT)

V) A bunch of Academic partners and Applied Research Labs

- **CEA-LIST / LLSP** (F) is developing a parallel design, programming and execution model to correctly manage the parallelism between applications, between tasks, and inside tasks. This includes parallelism expression, code generation, and execution framework to take advantage of multi-core or MPSoC
- **UVSQ** (F) (Université de Versailles Saint-Quentin-en-Yvelines) is working on the performance characterisation of multi-core / multithreaded systems and develops related tools for the optimisation and restructuring of applications.
- **INT** (F) (Institut National des Telecommunications) carries out research on parallelisation of applications, and is developing a prototype to transform an OpenMP program into an hybrid (OpenMP + MPI) program to exploit parallelism on distributed memory systems
- **UAB** (S) (Universitat Autonoma de Barcelona) carries out research and development of new software and hardware codesign methodologies for embedded systems. In ParMA, UAB focuses on interconnecting networks for Multiprocessors on a Chip (MPSoC) and on adapting MPI (the message passing library) to MPSoC environments
- **ROBOTIKER** (S) an R&D&I institution with knowledge in communication networks and reconfigurable hardware programming is studying the performance of MPSoCs in a network topology aided by discrete event simulators in order to select the most appropriate solution for implementing the next generation of Software Defined Radio.

- New architectures offer multilevel parallelism

  => Programming paradigms and tools must be developed to help restructure existing HPC applications so they fully benefit from this multilevel parallelism.

- Multi-core processors will be able to run a huge number of threads in parallel,

  => Dramatic improvements must be achieved in the way threads are allocated and monitored to minimise the time spent for exchanging data between threads.

  => Need more powerful, scalable, and user-friendly programming, debugging and performance analysis tools.

- Parallel architectures are coming in different forms – SMP, NUMA, MPSoC, …

  => Debugging and performance analysis tools must be able to run in these various contexts.

- Embedded systems will also be built with multi-core processors (MPSoC)

  => Multilevel parallel programming and execution models must be defined

  => Efficient interconnecting networks for MPSoC must be designed

- Each domain has its own characteristics and specific constraints

- Ensuite, montrer comment MAQAO permet d'identifier, à partir de l'analyse statique du code généré d'une part, et de la mesure des performances de la séquence de code asm obtenu en modifiant les options de compilation et/ou la façon d'écrire le code source correspondant.

Code Generation

PetaQCD, Orsay, 2009 January 20th-21st

CAPS Code Generation

1. CAPS in PetaQCD
2. HMPP in a nutshell
   - Directives for Hardware Accelerators (HWA)

§ HMPP Code Generation Capabilities
   - Code generation for GPU (CUDA, …)
   - Code generation for Cell (Available end Q1 2009)

§ Code Tuning
   - CAPS Tuner

Company Profile

- Founded in 2002
  - 25 staff
- Spin off of two research institutes
  - INRIA: processor microarchitecture and code generation techniques expertise
  - UVSQ: micro-benchmarking and architecture behavior expertise
- Mission: help our customers to efficiently build parallel applications
  - Integration services and expertise
  - Software tools (code generation and runtime)
- Customer references: Intel, Total, CEA, EADS, Bull, …
- R&D projects: POPS (System@tic), Milepost (IST), QCDNext (ANR) , PARA (ANR), …

CAPS in PetaQCD

- Provide code generation tools for Hardware Accelerators (HWA)
  - Highly optimized for LQCD
  - Based on a data parallel intermediate form (provided from a higher problem description language)
  - Mix of GPcore and HWA (hybrid parallelism)
- Provide iterative compilation techniques
  - Code tuning via optimization space exploration
- CAPS specific deliverables
  - D2.2 Decision/code generation process at run-time, and definition of a data parallel intermediate language to describe QCD applications
  - D2.3 Dynamic techniques for code generation


hmpp
hybrid multicore parallel programming

Simple C Example

To be executed on the HWA

```
#include <stdio.h>
#include <stdlib.h>

#pragma hmpp simple codelet, args[1].io=out
void simplefunc(int n, float v1[n], float v2[n], float v3[n], float alpha)
{
  int i;
  for (i = 0 ; i< n ; i++) {
    v1[i] = v2[i] * v3[i] + alpha;
  }
}
int main(int argc, char **argv) {
  unsigned int n = 400;
  float t1[400], t2[400], t3[400];
  float alpha = 1.56;
  unsigned int  j, seed = 2;
/* Initialization of input data*/
/* . . . */

#pragma hmpp simple callsite
  simplefunc(n,t1,t2,t3,alpha);

  printf("%f %f (...) %f %f \n", t1[0], t1[1], t1[n-2], t1[n-1]);
  return 0;
}
```

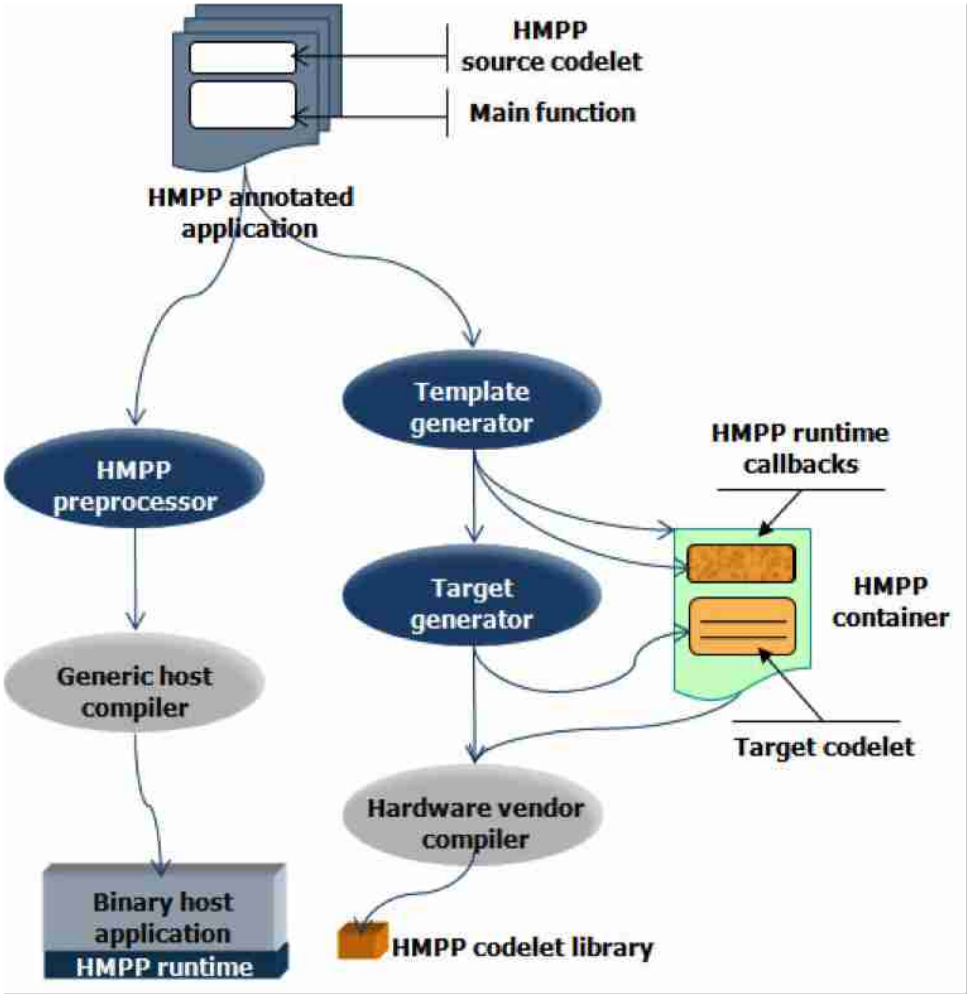
codelet / callsite directive set

Codelet Generation

Objectives

- Allow to transparently use HWA
  - From C or Fortran, Java to CUDA, Brook, ...
- Allow for code tuning at source code level
  - Directives based approach

Code Generation Flow



Codelet Generation

- C, Java or Fortran source code input
  - HWA oriented subset of the languages
- Set of directives to
  - Optimize target codelet generation
  - Express parallelism expression
- Make code tuning easier
- Generated code can also be tuned

Loop Parallelization

- Force or prevent the parallelization of loops
- Help defining kernels in a codelet

```
#pragma hmppcg parallel
for (i=0; i < n; i++) {
#pragma hmppcg noParallel
    for (j=0; j < n; j++) {
          D[i][j] = A[i][j] * E[3][j];
     }
}
```

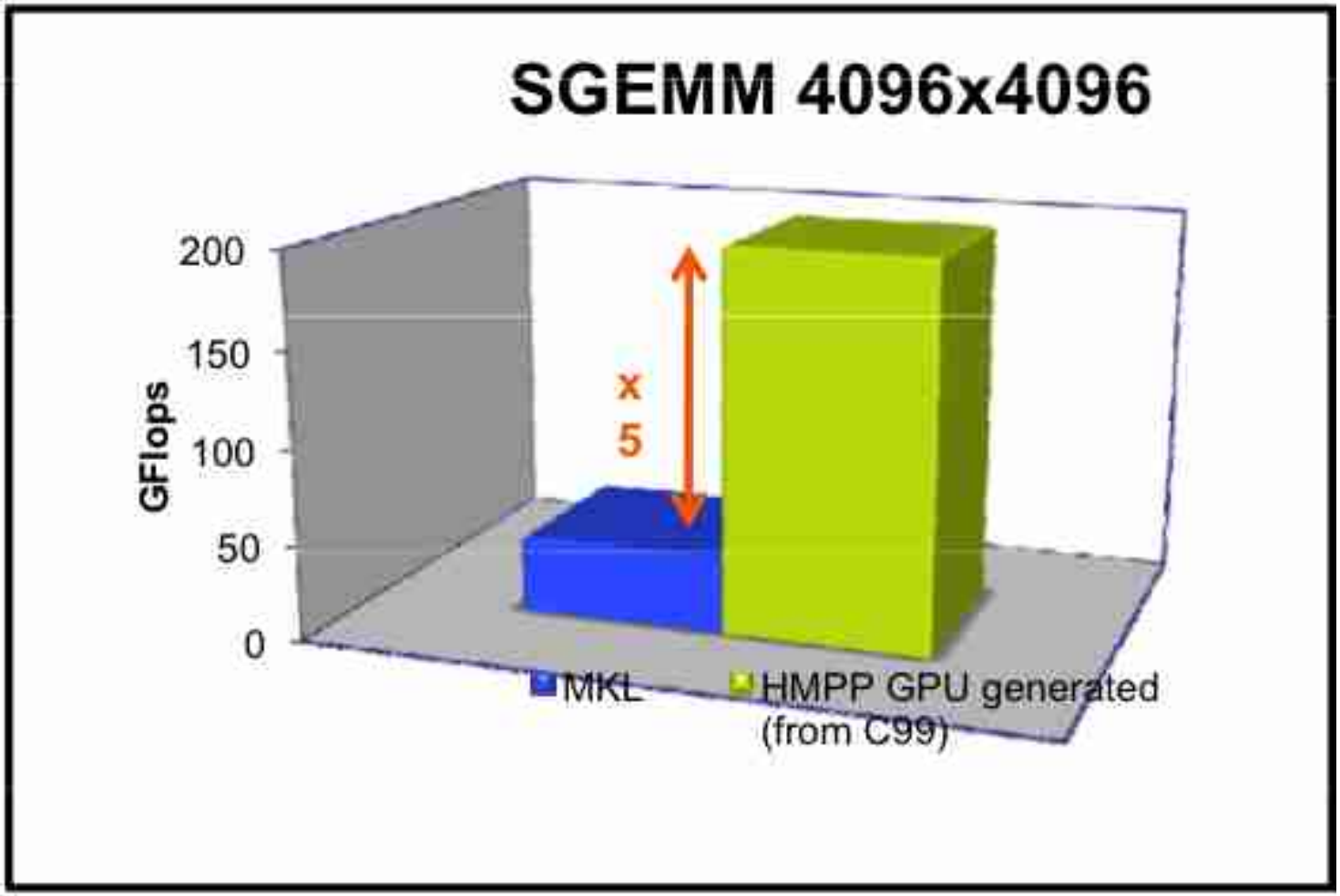
Input C Code Example 1

```
typedef struct{    float r, i;} Complex;
#pragma hmpp convolution2d codelet, args[data; opx].io=in, args[convr].io=out, target=CUDA
void convolution2d( Complex *data, int nx, int ny,  Complex *opx,
                    int oplx, int oply, Complex *convr) {
int hoplx = (oplx+1)/2;  int hoply = (oply+1)/2;
int iy, ix;
#pragma hmppcg parallel
for (iy = 0; iy < ny; iy++) {
#pragma hmppcg parallel
  for (ix = 0; ix < nx; ix++) {
    float dumr =0.0, dumi = 0.0;      int ky;
    for(ky = -(oply - hoply - 1); ky <= hoply; ky++) {
      int kx;
      for(kx = -(oplx - hoplx - 1); kx <= hoplx; kx++){
        int dx = min( max(ix+kx, 0), (nx - 1) );
        int dy = min( max(iy+ky, 0), (ny - 1) );
        dumr += data[dy * nx + dx].r * opx[(hoply - ky) * oplx + (hoplx - kx)].r;
        dumr -= data[dy * nx + dx].i * opx[(hoply - ky) * oplx + (hoplx - kx)].i;
        dumi += data[dy * nx + dx].r * opx[(hoply - ky) * oplx + (hoplx - kx)].i;
        dumi += data[dy * nx + dx].i * opx[(hoply - ky) * oplx + (hoplx - kx)].r;
       }
     }
     convr[iy*nx+ix].r = dumr; convr[iy*nx+ix].i = dumi;
   }
  }
}
```

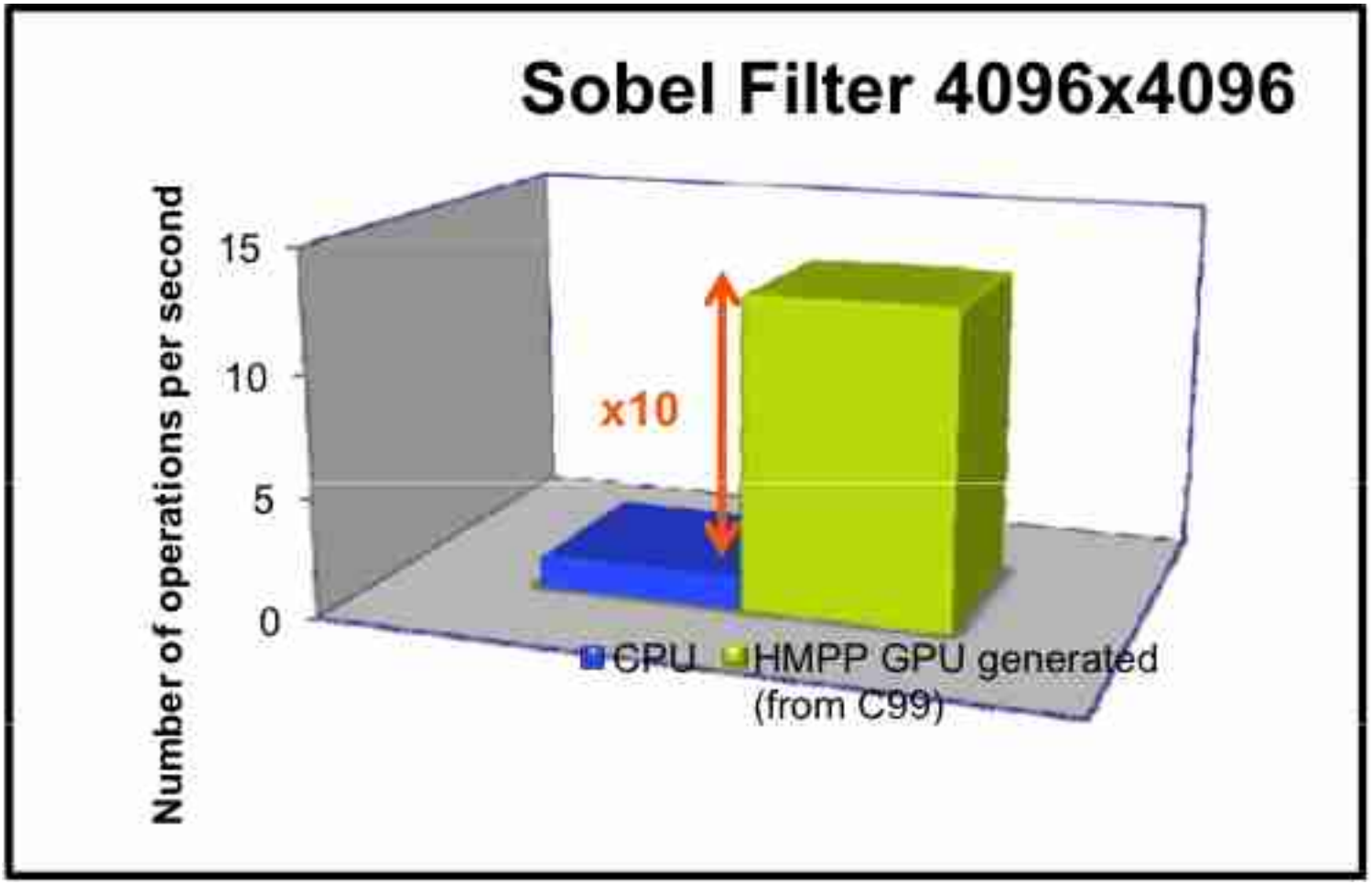
Input Fortran Code Example 2

```
!$HMPP sgemm3 codelet, target=CUDA, args[vout].io=inout
SUBROUTINE sgemm(m,n,k2,alpha,vin1,vin2,beta,vout)
INTEGER, INTENT(IN)    :: m,n,k2
REAL,   INTENT(IN)    :: alpha,beta
REAL,    INTENT(IN)    :: vin1(n,n), vin2(n,n)
REAL,    INTENT(INOUT) :: vout(n,n)
REAL     :: prod
INTEGER  :: i,j,k
!$HMPPCG unroll(8), jam(2), noremainder
!$HMPPCG parallel
DO j=1,n
    !$HMPPCG unroll(8), splitted, noremainder
    !$HMPPCG parallel
    DO i=1,n
        prod = 0.0
        DO k=1,n
          prod = prod + vin1(i,k) * vin2(k,j)
        ENDDO
        vout(i,j) = alpha * prod + beta * vout(i,j) ;
     END DO
END DO
END SUBROUTINE sgemm
```

MxM Performance



Performance Examples



Codelet Tuning

Codelet Tuning (1)

- Based on CAPSTuner Technology
  - Platform independent
- Iterative compilation technique to explore the optimization space
  - Explore source code transformations
    - Via a set of code transformation directives such as unroll-and-jam
  - Explore compiler options
  - Store the performance data in a well defined repository

Code Tuning (2)