Programming Multicore Challenges

François Bodin

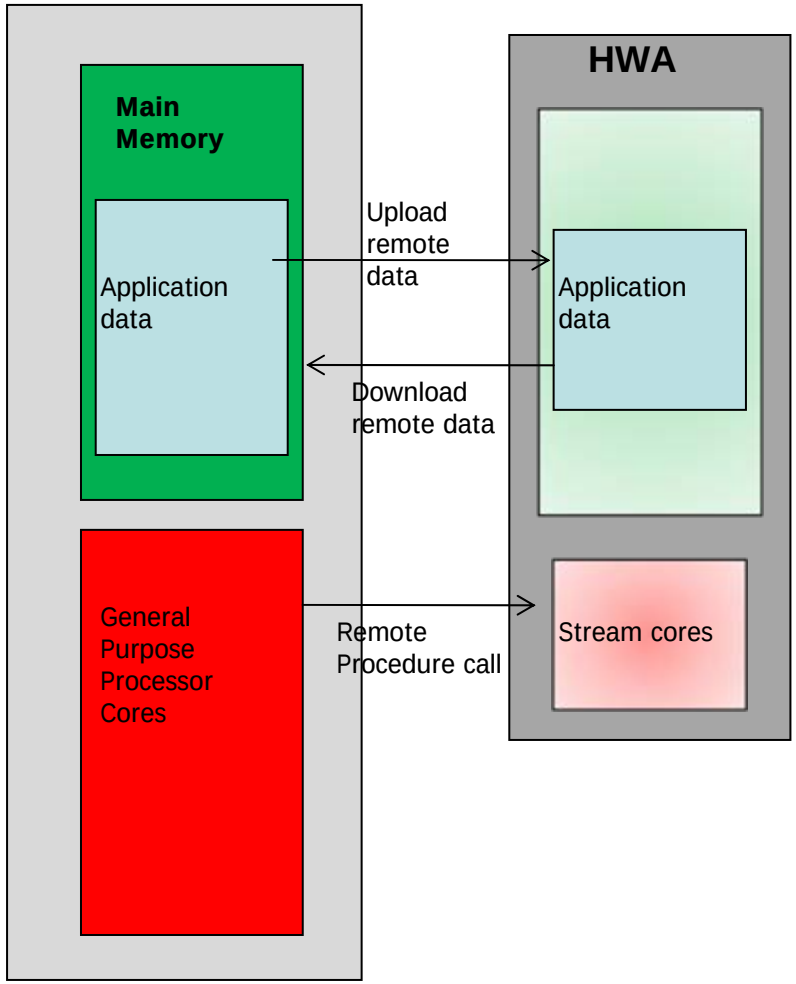
Irisatech, May 19th, 2009

Introduction

- Main stream applications will rely on new multicore / manycore architectures
  - It is about performance not parallelism
- Various heterogeneous hardware
  - General purpose cores
  - Application specific cores – GPU
- HPC and embedded applications are increasingly sharing characteristics

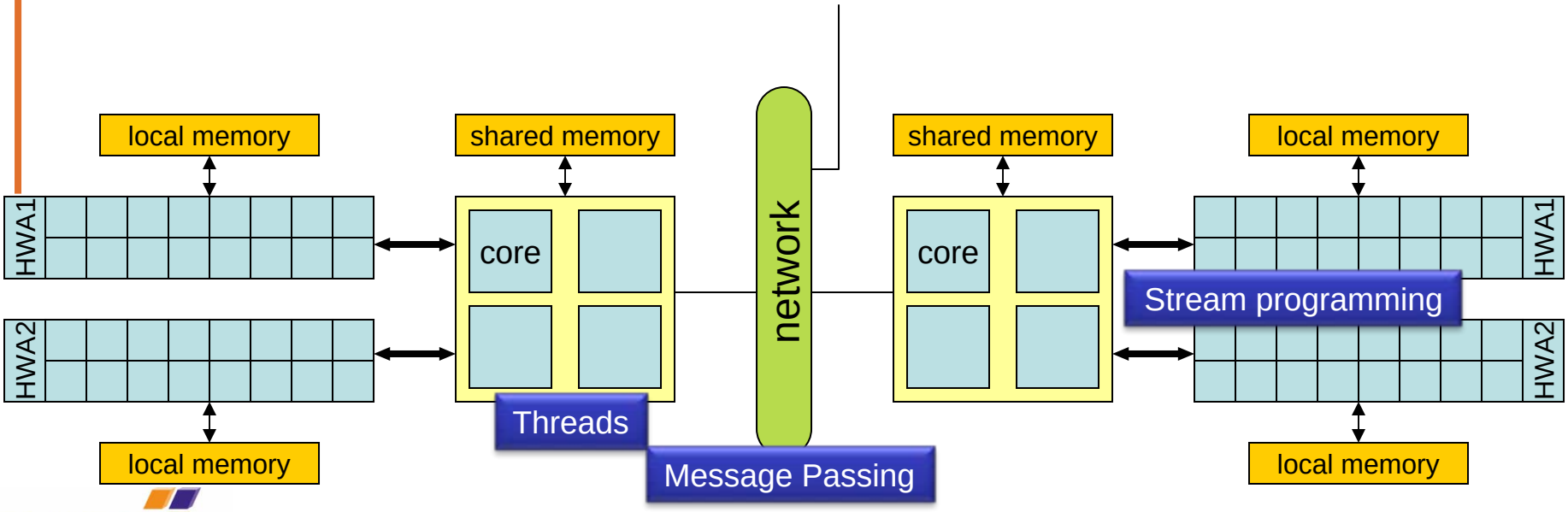
Manycore Architectures

- General purpose cores
  - Share a main memory
  - Core ISA provides fast SIMD instructions
- Streaming engines / DSP / FPGA
  - Application specific architectures ("*narrow band*")
  - Vector/SIMD
  - Can be extremely fast
- Hundreds of GigaOps
  - But not easy to take advantage of
  - One platform type cannot satisfy everyone
- Operation/Watt is the efficiency scale



Multiple Parallelism Levels

- Amdahl's law is forever, all levels of parallelism need to be exploited
  - Hybrid parallelism needed
- Programming various hardware components of a node cannot be done separately



Programming Multicores/Manycores

- Physical architecture oriented
  - Shared memory architectures
    - OpenMP, CILK, TBB, automatic parallelization, vectorization...
  - Distributed memory architectures
    - Message passing, PGAS (Partition Global Address Space), ...
  - Hardware accelerators, GPU
    - CUDA, OpenCL, Brook+, HMPP, ...
- Different styles
  - Libraries
    - MPI, pthread, TBB, SSE intrinsic functions, ...
  - Directives
    - OpenMP, HMPP, ...
  - Language constructs
    - UPC, Cilk, Co-array Fortran, UPC, Fortress, Titanium, ...

The Past of Parallel Computing, the Future of Manycore?

- The Past
  - Scientific computing focused
  - Microprocessor or vector based, homogeneous architectures
  - Trained programmers willing to pay effort for performance
  - Fixed execution environments
- The Future
  - New applications (multimedia, medical, ...)
  - Thousands of heterogeneous systems configurations
  - Unfriendly execution environments

Multi (languages) programming

- Happens when programmers needs to deal with multiple programming language
  - E.g. Fortran and Cuda, Java and OpenCL, ...
- Multiprogramming impacts on
  - Programmer's expertise
  - Program maintenance and correctness
  - Long term technology availability
- Performance programming versus domain specific programming
  - Libraries, parallel components to be provided to divide the issues

Can the Hardware be Hidden?

- Programming style is usually hardware independent but
  - Programmers needs to take into account available hardware resources
- *Quantitative* decisions as important as parallel programming
  - Performance is about quantity
  - Tuning is specific to a configuration
- Runtime adaptation is a key feature
  - Algorithm, implementation choice
  - Programming/computing decision

Manycore = Numerous Configurations

- Heterogeneity brings a lot of configurations

  Proc. x Nb Cores x HWA x Mem. Sys.

  =

  1000^s of configurations

- Code optimization strategy may differ from one configuration to another

Is it possible to make a single (a few) binary that will run efficiency on a large set of configurations?

Asymmetric Behavior Issue

- Cannot assume that all cores with same ISA provide equal performance
  - Core frequency/voltage throttling can change computing speed of some cores
    - e.g. Nehalem “turbo mode”
  - Simple (in order) versus complex (out-of-order) cores
  - Data locality effects
  - ...

How to deal with non homogeneous core behavior?

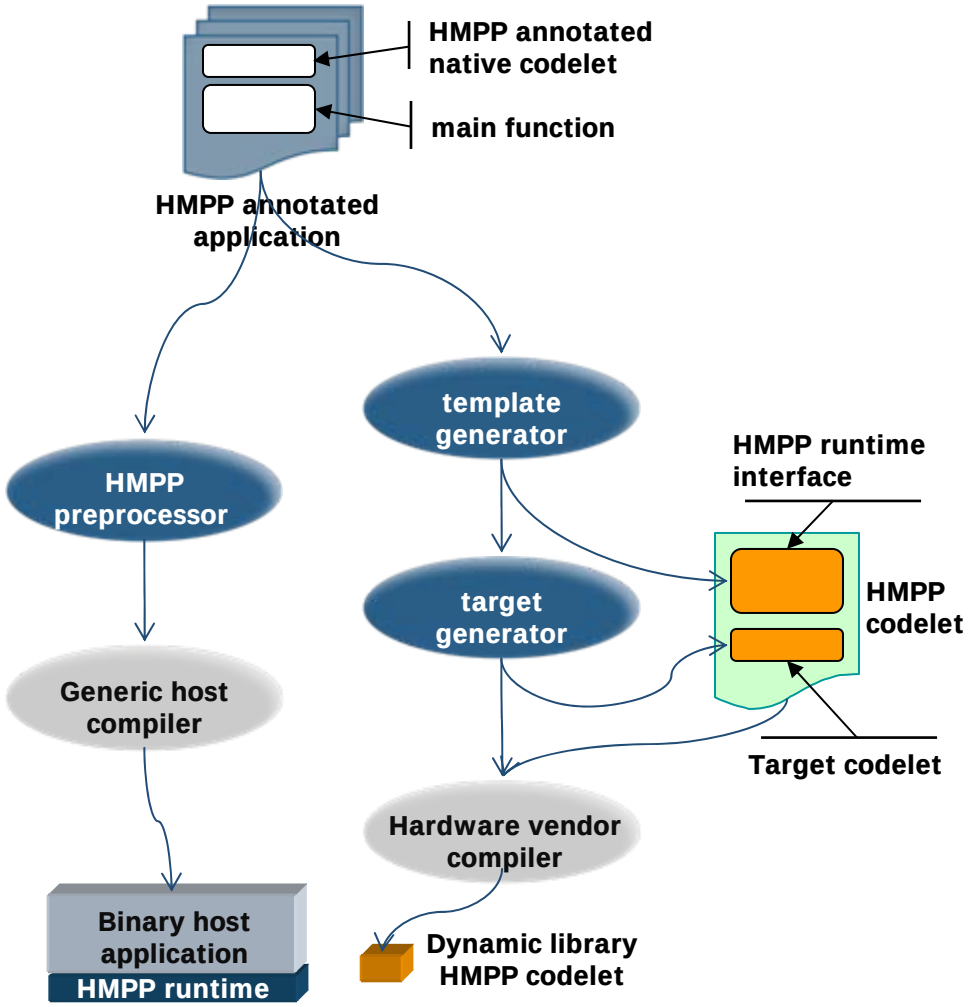
Manycore = Multiple μ -Architectures

- Each μ -architecture requires different code generation/optimization strategies
  - Not one compiler in many cases
- High performance variance between implementations
  - ILP, GPCore/TLP, HWA
- Dramatic effect of tuning
  - Bad decisions have a strong effect on performance
  - Efficiency is very input parameter dependent
  - Data transfers for HWA add a lot of overheads

How to organize the compilation flow?

CAPS Compiler Flow for Heterogeneous Targets

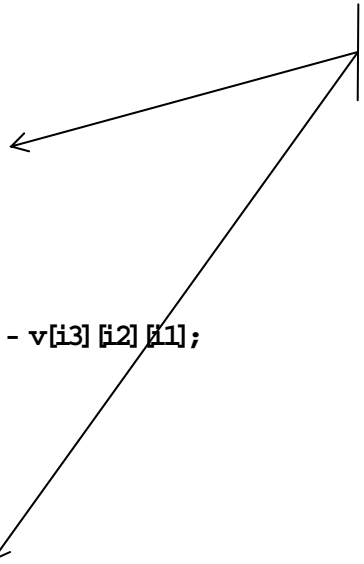
- Dealing with various ISAs
- Not all code generation can be performed in the same framework



Tuning Issue Example

```
#pragma hmpp astex_codelet__1 codelet &
#pragma hmpp astex_codelet__1 , args[c].io=in &
#pragma hmpp astex_codelet__1 , args[v].io=inout &
#pragma hmpp astex_codelet__1 , args[u].io=inout &
#pragma hmpp astex_codelet__1 , target=CUDA &
#pragma hmpp astex_codelet__1 , version=1.4.0
void astex_codelet__1(float u[256][256][256], float v[256][256][256], float c[256][256][256],
                      const int K, const float x2){
 astex_thread_begin:{
  for (int it = 0 ; it < K ; ++it){
   for (int i2 = 1 ; i2 < 256 - 1 ; ++i2){
    for (int i3 = 1 ; i3 < 256 - 1 ; ++i3){
     for (int i1 = 1 ; i1 < 256 - 1 ; ++i1){
      float coeff = c[i3][i2][i1] * c[i3][i2][i1] * x2;
      float sum = u[i3][i2][i1 + 1] + u[i3][i2][i1 - 1];
      sum += u[i3][i2 + 1][i1] + u[i3][i2 - 1][i1];
      sum += u[i3 + 1][i2][i1] + u[i3 - 1][i2][i1];
      v[i3][i2][i1] = (2. - 6. * coeff) * u[i3][i2][i1] + coeff * sum - v[i3][i2][i1];
     }
    }
   }
   for (int i2 = 1 ; i2 < 256 - 1 ; ++i2){
    for (int i3 = 1 ; i3 < 256 - 1 ; ++i3){
     for (int i1 = 1 ; i1 < 256 - 1 ; ++i1{

     .....
 }astex_thread_end:;
}
```



Need interchange
If aims at NVIDIA GPU

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(X),jam(2), noremainder
!$HMPPCG parallel
DO j=1,n
   !$HMPPCG unroll(X), 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
```

X>8 GPU compiler fails

X=8 200 Gigaflops

X=4 100 Gigaflops

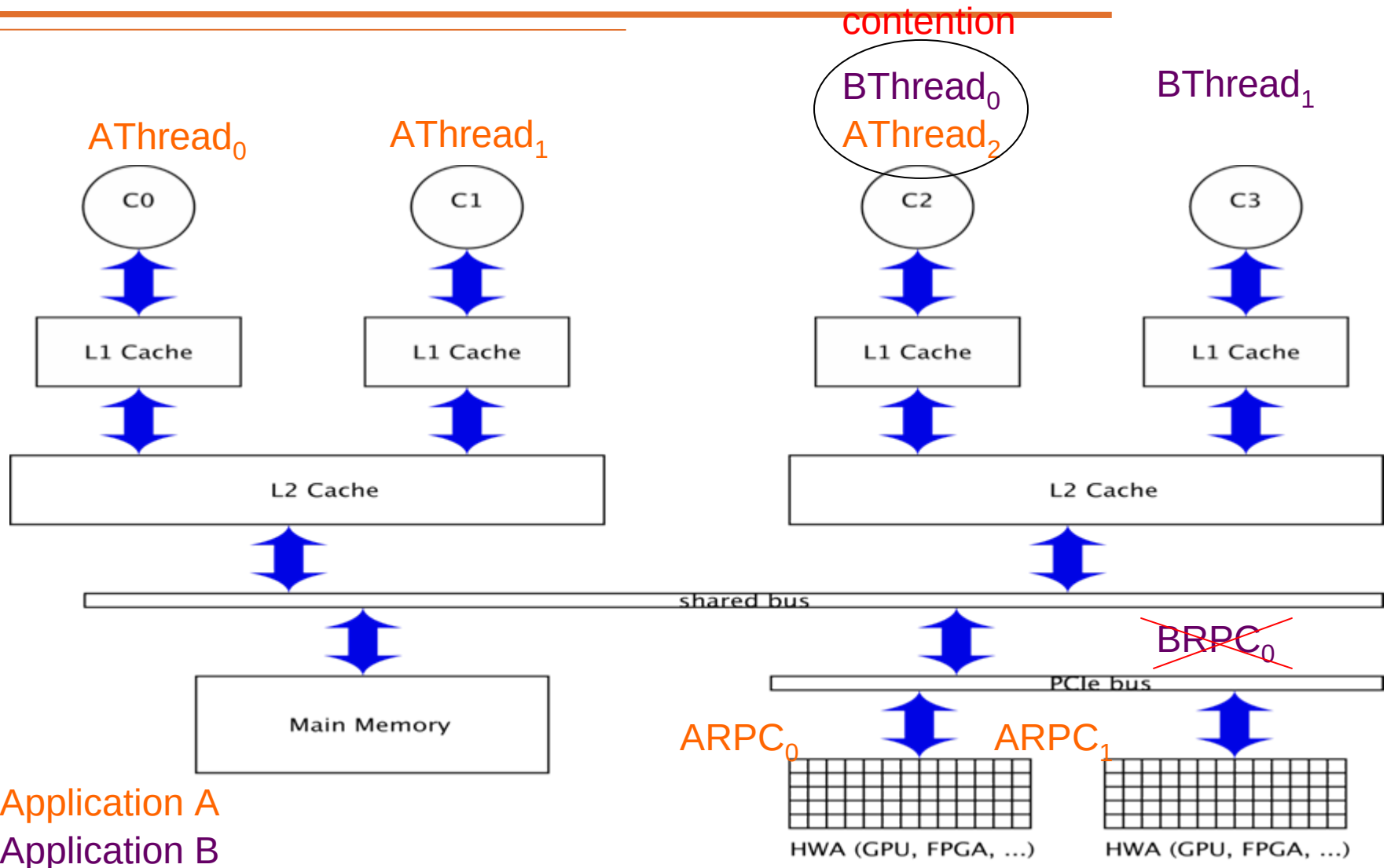
Multicore Workload

- Multiple applications sharing the hardware
  - Multimedia, game, encryption, security, health, ...
- Unfriendly environment with many competitions
  - Global resource allocation, no warranty on availability
  - Must be taken into account when programming/compiling
- Applications cannot always be recompiled
  - Most applications are distributed as binaries
- A binary will have to run on many platforms
  - Forward scalability or “write once, run faster on new hardware”
  - Loosing performance is not an option

Varying Available Resources

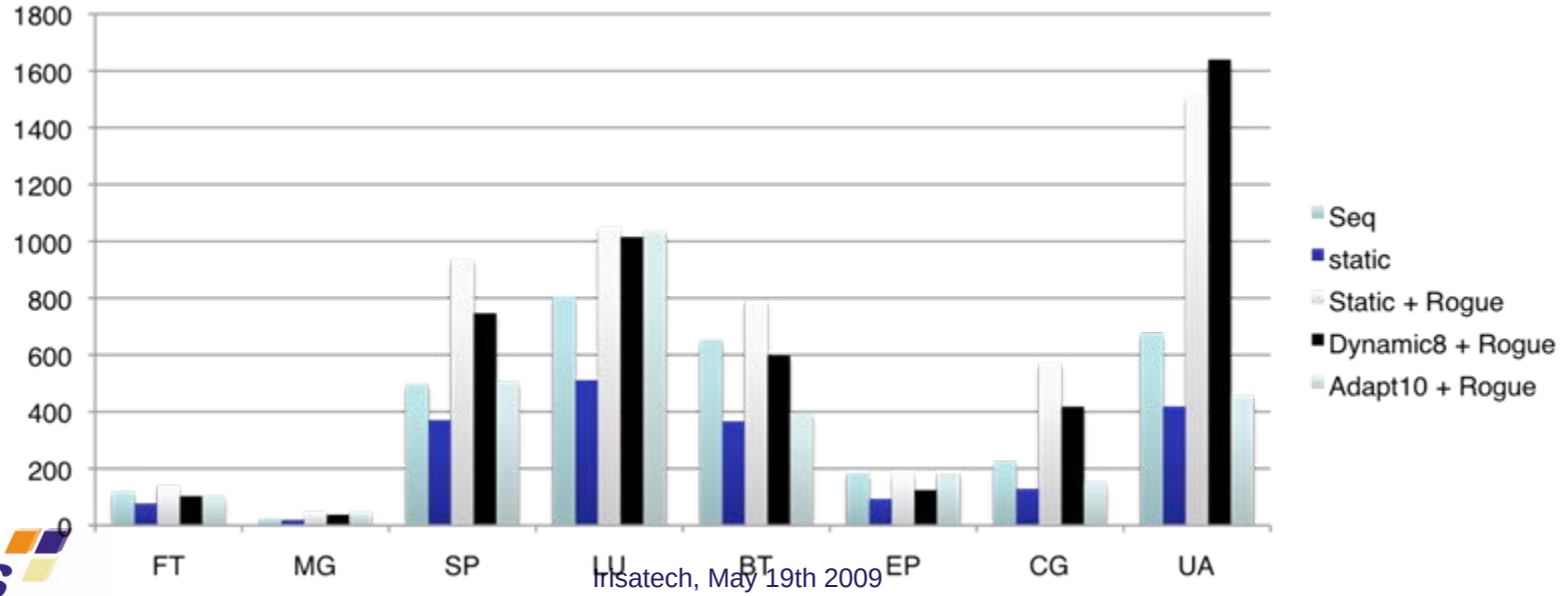
- Available hardware resources are changing over the execution time
  - e.g. a HWA may not be available
  - Data affinity must be respected
- How to avoid that conflict resource usage will not lead to global performance degradation?

Competition for Resources



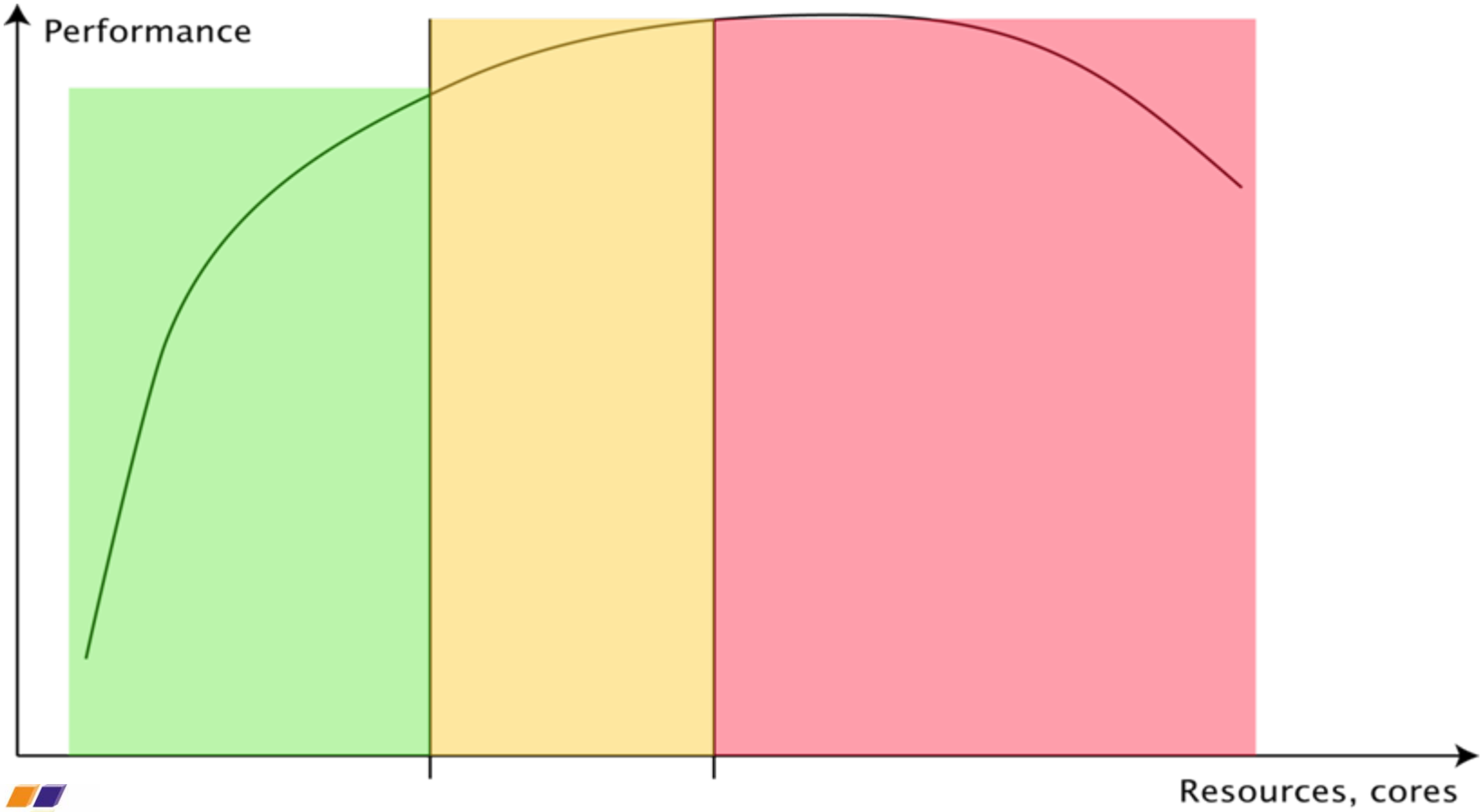
OpenMP in Unfriendly Environment

- OpenMP programs performance is strongly degraded when sharing resources
  - Example with NAS parallel benchmark, 2 cores, one *rogue* application using one of the core
  - Best loop scheduling strategy not identical on loaded or unloaded machine



Peak Performance is Not the Goal

- Maximizing the Return on Investment



Difficult Decisions Making with Alternative Codes (Multiversioning)

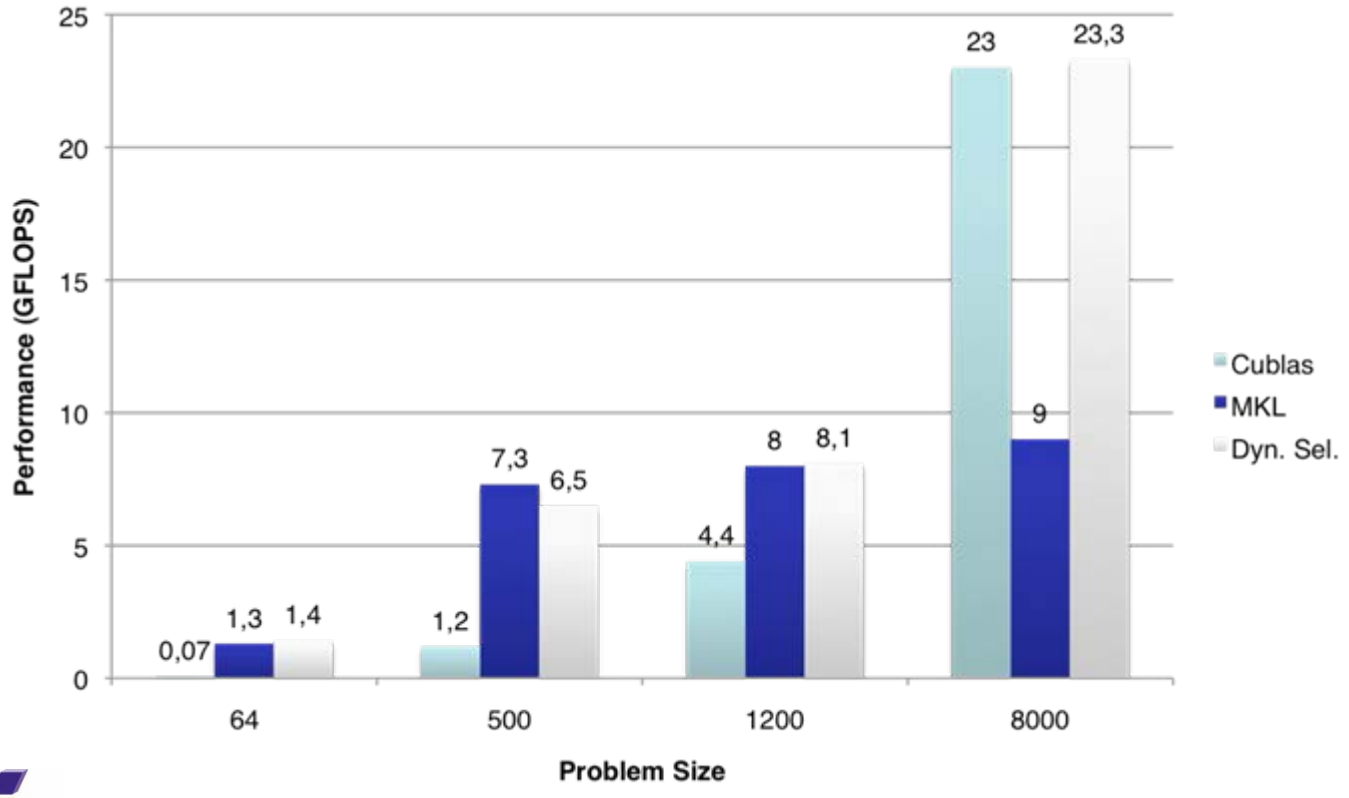
- Various implementations of routines are available or can be generated for a given target
  - CUBLAS, MKL, ATLAS, ...
  - SIMD instructions, GPcore, HWA, Hybrid
- No strict performance order
  - Each implementation has a different performance profile
  - Best choice depends on platform and runtime parameters
- Decision is a complex issue
  - How to produce the decision?

Illustrating Example: Dealing with Multiple BLAS Implementations

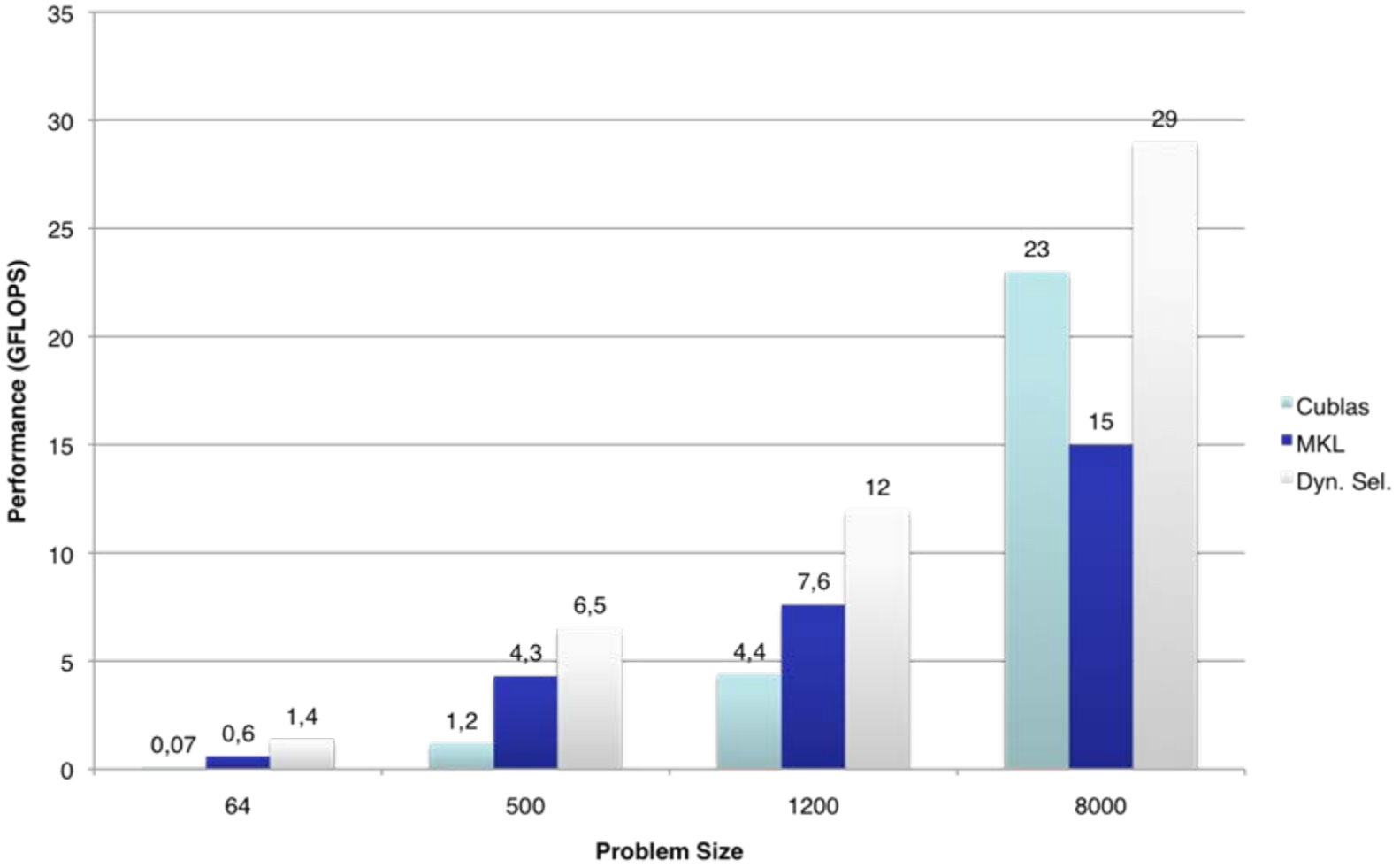
- Runtime selection of DGEMM in High Performance Linpack
  - Intel(R) Xeon(R) E5420 @ 2.50GHz
  - CUBLAS - Tesla C1060, Intel MKL
- Three binaries of the application
  - Static linking with CUBLAS
  - Static linking with MKL
  - Library mix with selection of routine at runtime
    - Automatically generated using CAPS tooling
- Three hardware resource configurations
  - GPU + 1, 2, and 4 cores used for MKL

Performance Using One Core

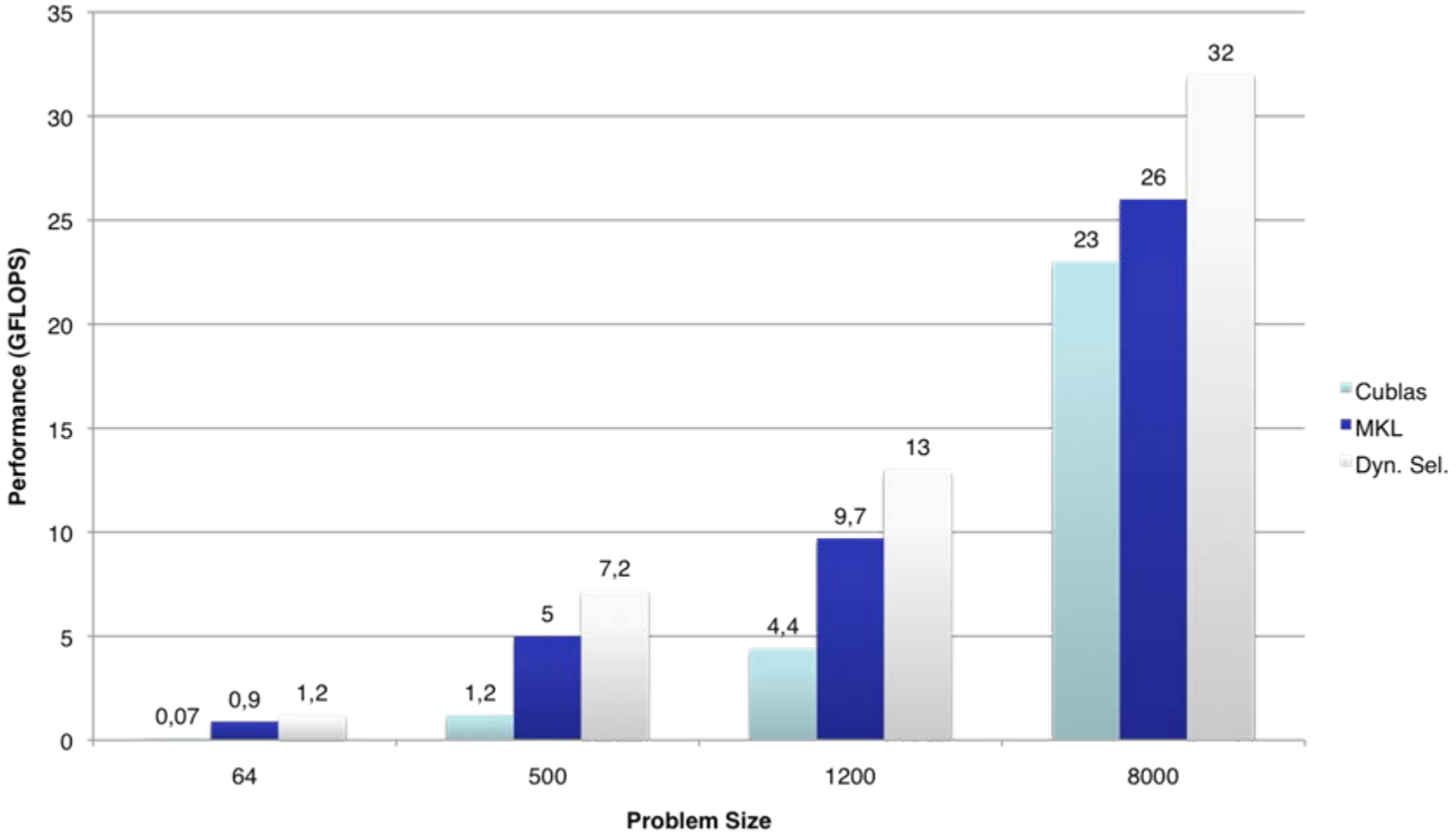
- Performance in Gigaflops
- 4 problem sizes: 64, 500, 1200, 8000



Performance Using Two Cores

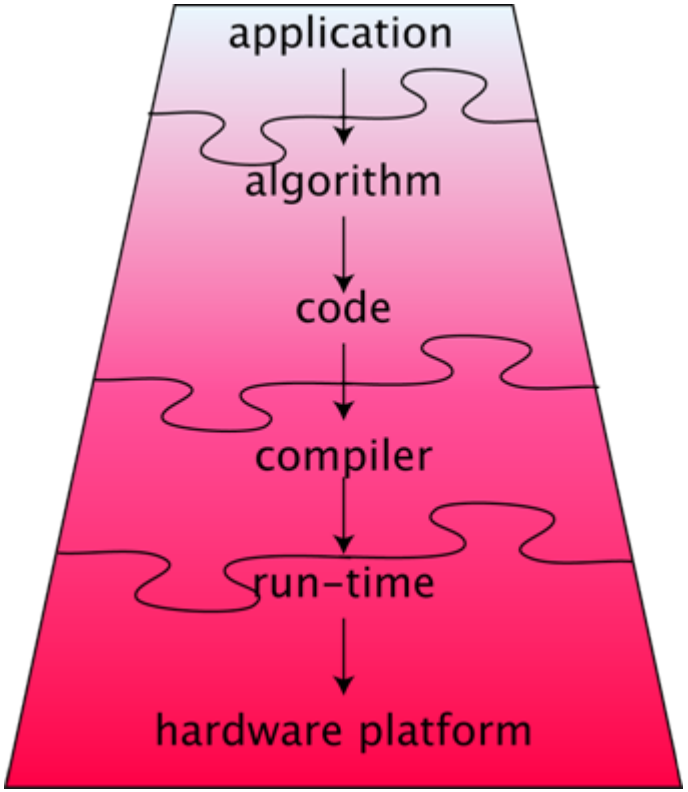


Performance Using Four Cores



The Challenges

- Programming
  - Medium
- Resources management
  - Medium
- Application deployment
  - Hard
- Portable performance
  - Extremely hard



Research Directions

- New Languages
  - X10, Fortress, Chapel, PGAS languages, OpenCL, MS Axum, ...
- Libraries
  - Atlas, MKL, Global Array, Spiral, Telescoping languages, TBB, ...
- Compilers
  - Classical compiler flow needs to be revisited
  - Acknowledge lack of static performance model
  - Adaptative code generation
- OS
  - Virtualization/hypervisors
- Architectures
  - Integration on the chip of the accelerators
    - AMD Fusion, ...
  - Alleviate data transfers costs
    - PCI Gen 3x

Key for the short/mid term

Directives Based Parallel Programming

- Directives
  - Do not need a new programming language
  - Already in state of the art (e.g. OpenMP)
  - Keep incremental development possible
  - Avoid exit cost
- Does not address large scale parallelism
  - But this is not (yet) the issue for multicore nodes
- Path chosen by CAPS entreprise
  - Heterogeneous Multicore Parallel Programming

Conclusion

- Multicore ubiquity is going to have a large impact on software industry
  - New applications but many new issues
- Will one parallel model fit all?
  - Surely not but multi languages programming should be avoided
- Toward Adaptative Parallel Programming
  - Compiler alone cannot solve it
  - Compiler must interact with the runtime environment
  - Programming must help expressing global strategies / patterns
  - Compiler as provider of basic implementations
  - Offline-Online compilation has to be revisited