CAPS

Innovative tools for a new paradigm

Parallel Hybrid Computing

Stéphane Bihan, CAPS

SYSTEM@TIC PARIS-REGION

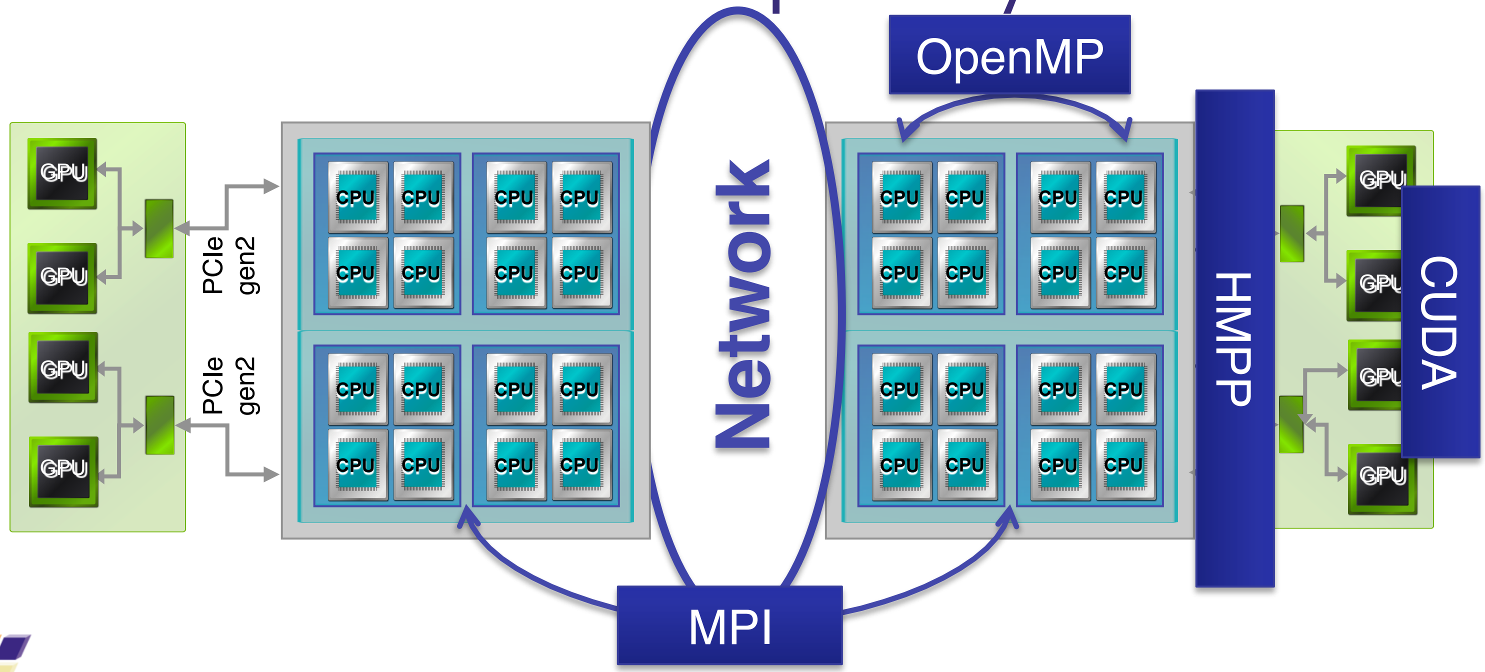
POPS PetaOpérations Par Seconde

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 – GPUs (HWAs)
- HPC and embedded applications are increasingly sharing characteristics

Multiple Parallelism Levels

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



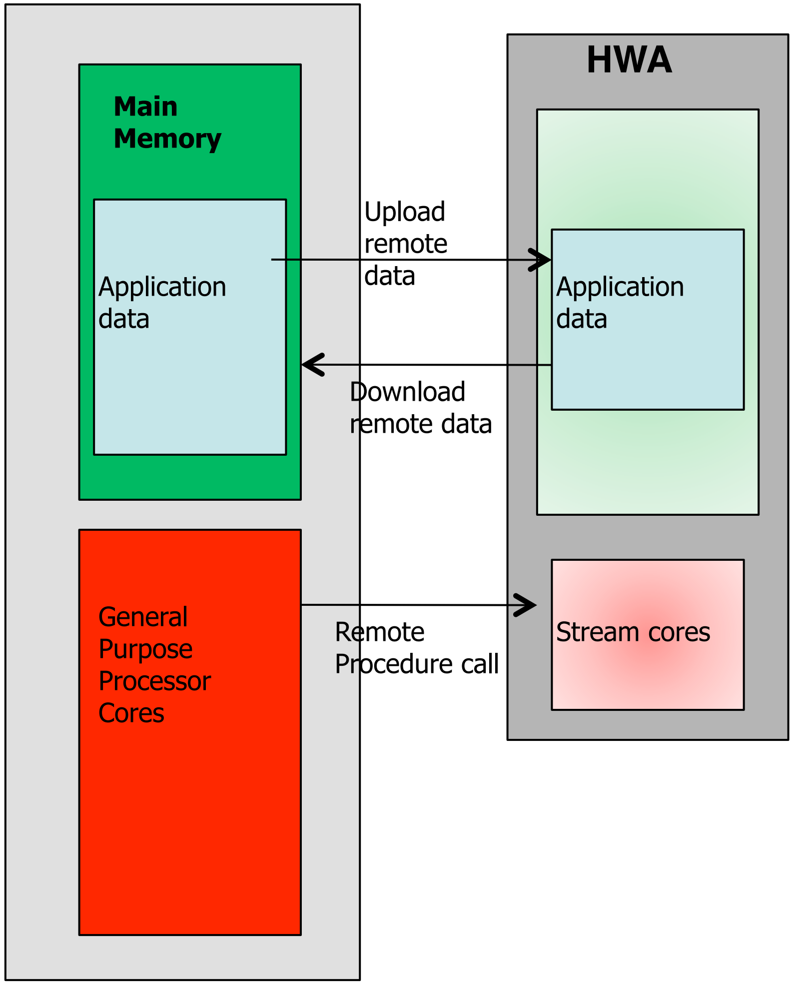
Programming Multicores/ Manycores

- 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, …

An Overview of Hybrid Parallel Computing

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



The Past of Parallel Computing, the Future of Manycores?

- 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

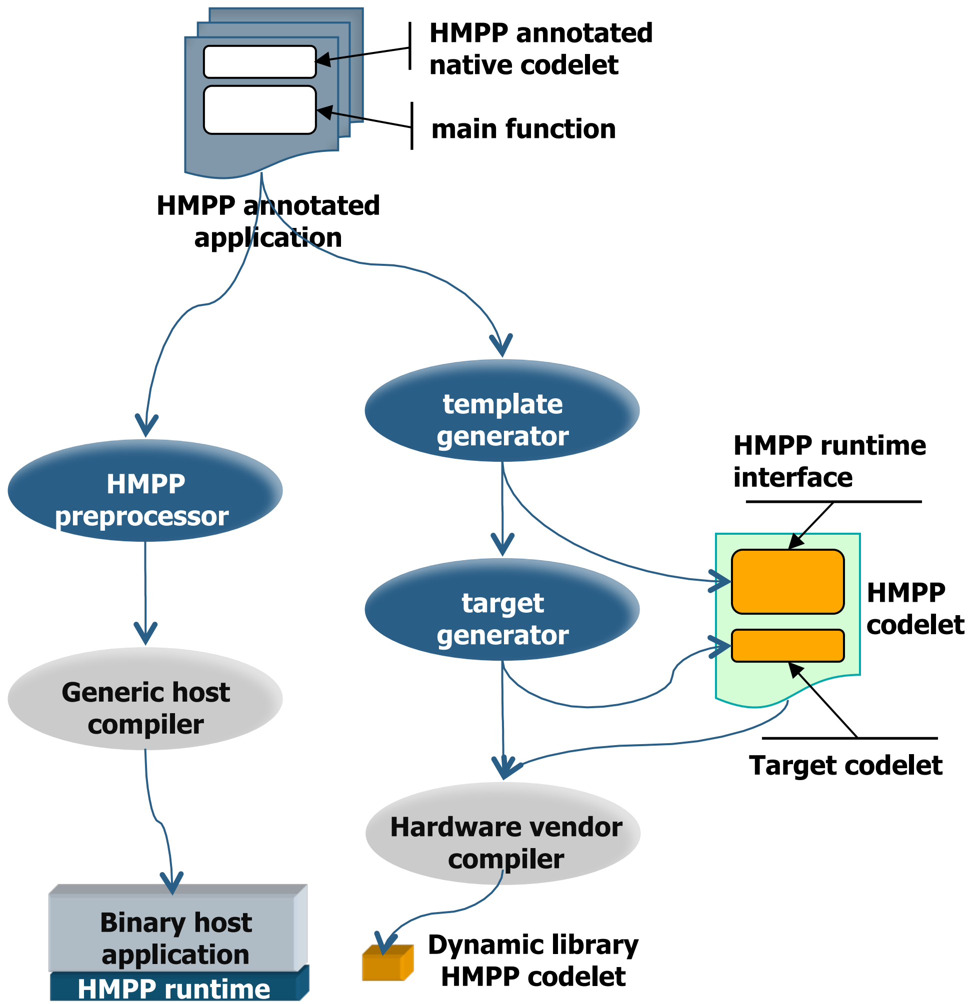
Manycore = Multiple µ-Architectures

- 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



HMPP Approach

HMPP Objectives

- Efficiently orchestrate CPU/GPU computations in legacy code
  - With OpenMP-like directives
- Automatically produce tunable manycore applications
  - C and Fortran to CUDA data parallel code generator
  - Make use of available compilers to produce binary
- Ease application deployment

HMPP…
a high level abstraction for manycore programming

HMPP1.5 Simple Example

```
#pragma hmpp sgemmlabel codelet, target=CUDA, args[vout].io=inout
extern void sgemm( int m, int n, int k, float alpha,
                   const float vin1[n][n], const float vin2[n][n],
                   float beta, float vout[n][n] );


int main(int argc, char **argv) {
…
 for( j = 0 ; j < 2 ; j++ ) {
#pragma hmpp sgemmlabel callsite
    sgemm( size, size, size, alpha, vin1, vin2, beta, vout );
 }
```

```
#pragma hmpp label codelet, target=CUDA:BROOK, args[v1].io=out
#pragma hmpp label2 codelet, target=SSE, args[v1].io=out, cond="n<800"
void MyCodelet(int n, float v1[n],  float v2[n], float v3[n])
{ int i;
  for (i = 0 ; i < n ; i++) {
    v1[i] = v2[i] + v3[i];
  }
}
```

Group of Codelets (HMPP 2.0)

- Declare group of codelets to optimize data transfers
- Codelets can share variables
  - Keep data in GPUs between two codelets
  - Avoid useless data transfers
  - Map arguments of different functions in same GPU memory location (`equivalence` Fortran declaration)

Flexibility and Performance

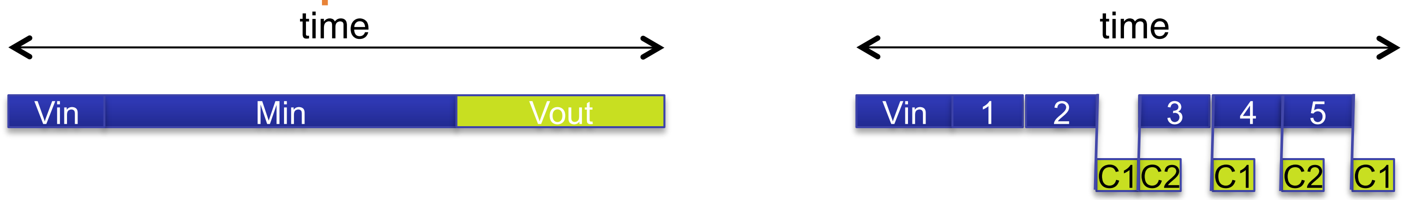
CAPS

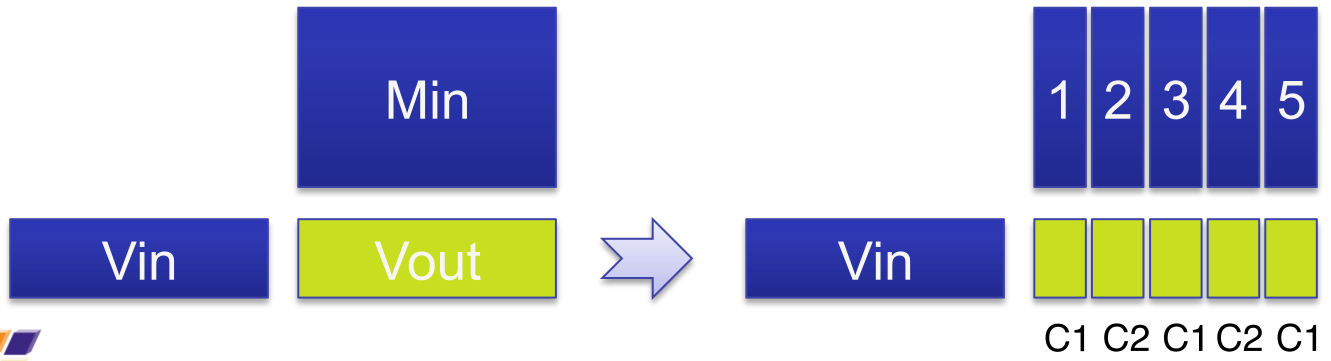
Optimizing Communications

- Exploit two properties
  - Communication / computation overlap
  - Temporal locality of parameters
- Various techniques
  - Advancedload and Delegatedstore
  - Constant parameter
  - Resident data
  - Actual argument mapping

Hiding Data Transfers

- Pipeline GPU kernel execution with data transfers
  - Split single function call in two codelets (C1, C2)
  - Overlap with data transfers





Advancedload Directive

- Avoid reloading constant data

```
int main(int argc, char **argv) {
…
#pragma hmpp simple advancedload, args[v2], const
  for (j=0; j<n; j++){
#pragma hmpp simple callsite, args[v2].advancedload=true
     simplefunc1(n,t1[j], t2, t3[j],  alpha);
    }
#pragma hmpp label release
…
}
```

`t2` is not reloaded each loop iteration

Actual Argument Mapping

- Allocate arguments of various codelets to the same memory space
  - Allow to exploit reuses of argument to reduce communications
  - Close to `equivalence` in Fortran

```
#pragma hmpp <mygp> group, target=CUDA
#pragma hmpp <mygp> map,   args[f1::inm; f2::inm]

#pragma hmpp <mygp> f1 codelet, args[outv].io=inout
static void matvec1(int sn, int sm,
                    float inv[sn], float inm[sn][sm], float outv[sm])
{
  ...
}
#pragma hmpp <mygp> f2 codelet, args[v2].io=inout
static void otherfunc2(int sn, int sm,
                       float v2[sn], float inm[sn][sm])
{
  ...
}
```

Arguments share the same space on the HWA

HMPP Tuning

```
!$HMPP sgemm3 codelet, target=CUDA, args[vout].io=inout
SUBROUTINE sgemm(m,n,k2,alpha,vin1,vin2,beta,vout)
INTEGER, INTENT(IN)    :: m,n,k2
REAL,   INTENT(IN)    :: alpha,beta
REAL,    INTENT(IN)    :: vin1(n,n), vin2(n,n)
REAL,    INTENT(INOUT) :: vout(n,n)
REAL     :: prod
INTEGER  :: i,j,k
!$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

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
  - Directive based programming is a safe approach
  - Ideally OpenMP will be extended to HWA
- 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