HMPP Workbench

Directive-based Programming Tools for Manycore Architectures
Outline

• Issues of Hybrid Programming
• HMPP Features
• When Should HMPP Be Used?
• HMPP Programming Tools: What Is DevDeck?
Issues of Hybrid Programming
What is Hybrid Computing?

• Various heterogeneous hardware
  o General purpose cores
  o Application specific cores (e.g. GPUs)

• Mainstream HPC applications will rely on these multicore / manycore hybrid architectures

• Applications need to be ported to new hardware
  o This hardware may change along the application’s life
    • FPGA ➔ GPU ➔ ?
  o Can be extremely fast
  o But not easy to leverage
  o Restriction to one platform is not acceptable
Parallelism and Architectures

General purpose architecture

Massively data parallel

Needs 1000s of computation threads to be efficient
Offloading Computations

- **From the host**
  - General purpose cores
  - Share a main memory

- **To the device**
  - GPU, DSP, FPGA...
  - Streaming cores (scalar, vector, SIMD)
  - Application specific architectures

- **Software should stay hardware-independent**
  - New architectures / languages to master (today mainly GPUs)
  - Hybrid solutions evolve ➔ redo the work each time it changes
  - Directives are a good solution
High Level Hybrid Programming

- **C & Fortran programming directives**
  - Preserve legacy code

- **A compiler integrating GPU stream code generators**
  - Insulate hardware specific implementations

- **Runtime library**
  - Ease application deployment
HMPP Application Execution

- HMPP handles identified compute intensive sections of code
  - Called codelets
  - Designed to be offloaded onto specific targets
  - HMPP targets GPUs as hardware accelerators

- Manages both applications and hardware
  - Distributes computations according to target-specific requests and available hardware

- Detects presence of and available GPUs
  - Ease application deployment on heterogeneous configurations
  - Falls back to native version when failing
HMPP Workbench

- **Source-to-source compiler**
  - Takes HMPP-annotated application as input
  - Automatically generates hardware-specific codes as output
- **And tools to automatically build hybrid applications**
- **Ease application development with directives**
  - Low learning curve
  - Ease readability and maintenance
  - Preserve legacy code
  - No exit cost
- **Ease application deployment**
  - For hardware portability
  - Aims at improving performance portability
- **Complementary with other programming tools**
  - Compliant with parallel tools (OpenMP/MPI), debuggers, profilers…
Offloading Computations with HMPP

```c
#include <stdio.h>
#include <cuda_runtime.h>

#define SIZE 64

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

**CODELET**: Identify a pure function/subroutine to accelerate

**CALLSITE**: Execute the CODELET
HMPP Compilation Tools

- HMPP generates hardware-specific codes
- HMPP comes with building tools to drive all compilation passes
  - HMPP is a compiler driver
- Two main passes
  - Host application compilation
  - Codelet production
HMPP Directives

• HMPP gives control over transfers and computations to the user:
  - CODELET: Specialize a subroutine
  - CALLSITE: Specialize a call statement
  - ACQUIRE: Acquire HWA
  - RELEASE: Release HWA
  - ALLOCATE: Allocate memory on the HWA
  - FREE: Free memory on the HWA
  - ADVANCEDLOAD: Explicit data transfer CPU -> HWA
  - DELEGATEDSTORE: Explicit data transfer HWA -> CPU
  - SYNCHRONIZE: Wait for completion of the callsite
  - GROUP: Define a group of codelets
#pragma hmpp sgemm codelet, target=CUDA, transfer[*]=atcall
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 sgemm callsite
        sgemm( size, size, size, alpha, vin1, vin2, beta, vout );
    }
    ...

With only two directives!
#pragma hmpp sgemm codelet, target=CUDA, transfer[*]=manual
extern void sgemm( ... )

int main(int argc, char **argv) {
...
#pragma hmpp sgemm allocate, args[vin1;vin2;vout].size={size,size}
#pragma hmpp sgemm advancedload, args[vin1;vin2;vout;m;n;k;alpha;beta]

    for( j = 0 ; j < 2 ; j++ )
    {
        #pragma hmpp sgemm callsite, asynchronous, &
        sgemm( size, size, size, alpha, vin1, vin2, beta, vout );
        ...
        #pragma hmpp sgemm synchronize
    }

#pragma hmpp sgemm delegatedstore, args[vout]
#pragma hmpp sgemm release
Directives for Code Generation Tweaking

- By adding properties
  - (no)parallel, alias

- Using code transformation
  - Loop tiling, unroll, jam, permute, fuse, …

- Using target specific directives
  - Micro Architecture Management (warp size…)
  - Memory Management (CUDA shared memory, constant….)
#pragma hmp dgemm codelet, target=CUDA, args[C].io=inout
void dgemm( int n, double alpha, const double *A, const double *B, 
double beta, double *C ) {

int i;
int j;
int k;
double prod;
#pragma hmpccg(CUDA) grid blocksize "64x1"
#pragma hmpccg(CUDA) permute j,i
#pragma hmpccg(CUDA) unroll(8), jam, split, noremainder
for( i = 0 ; i < n; i++ ) {
#pragma hmpccg(CUDA) unroll(4), jam(i), noremainder
   for( j = 0 ; j < n; j++ ) {
      prod = 0.0f;
      for( k = 0 ; k < n; k++ )
         prod += A[i][k] * B[k][j];
      C[i][j]= alpha * prod + beta * C[i][j];
   }
}
}
Using Shared Memory

#define N (256*10+2*DIST)
void conv1(int A[N], int B[N])
{
    int i,k;
    int buf[DIST+256+DIST];
    int grid = 0;
    #pragma hmppcg set grid = GridSupport()
    if (grid){
        #pragma hmppcg grid blocksize 256x1
        #pragma hmppcg parallel
        for (i=DIST; i<N-DIST; i++){
            #pragma hmppcg grid shared buf
            int t;
            #pragma hmppcg set t = RankInBlock(i)
            // Load the first 256 elements
            buf[t] = A[i-DIST];
            // Load the remaining elements
            if (t < 2*DIST)
                buf[t+256] = A[i-DIST+256];
            #pragma hmppcg grid barrier
        }
    }
}
Generated Codelet Performance – advanced use

SGEMM Performance

![Graph showing performance of SGEMM](image-url)

- **Taille des matrices**: 64, 256, 448, 640, 832, 1024, 1216, 1408, 1600, 1792, 1984, 2176, 2368, 2560, 2752, 2944, 3136, 3328, 3520, 3712, 3904, 4096, 4288, 4480, 4672, 4864, 5056, 5248, 5440, 5632, 5824, 6016, 6208, 6400, 6592, 6784, 6976, 7168, 7360, 7552, 7744, 7936, 8128

- **Gflop/s**: 0, 100, 200, 300, 400, 500, 600, 700

- **Legend**:
  - Red: HMPP
  - Green: CUBLAS
  - Purple: MAGMA
Generated Codelet Performance – advanced use

DGEMM Performance

Taille des matrices

Gflop/s

HMPP
CUBLAS
MAGMA
HMPP Supported Hardware and Applications

- HMPP v2.5.5
  - Input code
    - C
    - Fortran 90
  - Targeted accelerators
    - CUDA (Nvidia)
    - OpenCL (AMD/ATI, Nvidia)
  - Targeted Operating Systems
    - Linux
    - Windows

- HMPP v3.0.3
  - Input code
    - C
    - Fortran 90
    - C++ (API)
  - Targeted accelerators
    - CUDA (Nvidia)
    - OpenCL (AMD/ATI, Nvidia)
  - Targeted Operating Systems
    - Linux
    - Windows (coming soon)
• HMPP is on-node, but can be used:
  o In OpenMP directives (or any pthread tech)
  o In MPI implementations
  o With OpenMP and MPI…
When Should HMPP Be Used?
HMPP Methodology

• HMPP offers an incremental, progressive approach for code porting
  o Simple programming ➔ Transfer optimization ➔ Codelet tuning
  o Fits well the recommended Porting Methodology

• Reduces development time

• Enables the user to reach its performance goal
  o High performances
  o Best performance / development time ratio

• CAPS provides tools for efficient code porting
  o DevDeck, CAPS Porting Methodology…
What Is DevDeck?
DevDeck: current configuration

- debugger
  - allinea ddt
    - the distributed debugging tool

- libraries
  - CULA tools

- HMPP Compiler
- HMPP Wizard
- HMPP Performance Analyzer
Allinea DDT – Debugging Tool for CPU/GPU

• Debug your kernels:
  o Debug CPU and GPU concurrently

• Examine thread data
  o Display variables
  o ...

• Integrated with HMPP
  o Allows HMPP directives breakpoints
  o Step into HMPP codelets
Offer different levels of information:

- Static:
  - Analysis of the source code
  - Take into account HMPP Code Generation directives

- Dynamic:
  - CPU profiling
  - Feedback on GPU execution with the Performance Analyzer

2 kinds of analysis

- Inside HMPP code: guide the optimization of the code (advice)
- Outside the HMPP code:
  - Provide information on potential parallel loops
  - Library usage opportunity
HMPP Wizard and Performance Analyzer

- CPU Profiling view
- Source files view
- Advice view
- GPU execution feedback view

Welcome to HMPP Wizard ➔ Home

Summary of the Wizard results

<table>
<thead>
<tr>
<th>Analysis Date</th>
<th>11/13/11 05:35:58</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source code languages</td>
<td>C(4)</td>
</tr>
<tr>
<td></td>
<td>Fortran(1)</td>
</tr>
<tr>
<td>Number of lines of code</td>
<td>1363</td>
</tr>
<tr>
<td>Number of analyzed functions</td>
<td>50</td>
</tr>
<tr>
<td>Files with a diagnosis</td>
<td>4 C file(s) 853 lines</td>
</tr>
<tr>
<td></td>
<td>1 Fortran file(s) 530 lines</td>
</tr>
</tbody>
</table>

Application | File | Functions |
-------------|------|-----------|
             | sample/data/src/mycode.c | initLoopAtoi |
             |                    | initL.aop    |
             |                    | loopUnrolled |
             |                    | loopStride2  |
             |                    | notParallel  |
             |                    | notPerfectlyNested |
             |                    | notPerfectlyNestedFix1 |
HMPP Wizard: CPU Profile view

- Detect CPU hot spots
- Provide a CPU performance overview
Welcome to HMPP Wizard » Advice Results

Close this tab

88 double t_create1 = ctkRealTimer();
89 pr2c = fftw_plan_dft_r2c_lid(n, idata_real, odata_intermediate, FFTW_ESTIMATE);
90 po2r = fftw_plan_dft_c2r_lid(n, odata_intermediate, odata_real_GPU FFTW_ESTIMATE);
91 double t_create2 = ctkRealTimer();
92
93 double t_exec_pr2c1 = ctkRealTimer();
94 fftw_execute(pr2c);
95 double t_exec_pr2c2 = ctkRealTimer();
96
97 double t_filter1 = ctkRealTimer();
98 filter[n, (double _Complex *) odata_intermediate, cf];
99 double t_filter2 = ctkRealTimer();
100
101 double t_exec_pc2r1 = ctkRealTimer();

Detected potential issue

HMPP-ALT-FFT/VERSION1 /exec_D2Z_Z2D.c @line 95 - Advice54: A call to the standard FFTW function "fftw_execute" has been detected inside a function.

Advice

Consider using an optimized library for your application with the HMPP ALT proxy.

Loop Statistics

- Number of array access: 1
- Number of operations: 2 including 0 flops
- Number of intrinsic operations: 0 including 0 flops

For more details, click here

Advice

- The computation may fetch few...
Conclusion

• Standardization effort is going on
  o OpenCL from Khronos
  o OpenMP accelerator subcommittee
  o HMPP as an Open Standard

• Hardware cannot be totally hidden to programmers
  o e.g. exposed memory hierarchy
  o Efficient programming rules must be clearly stated

• Use HMPP for
  o Fast development
  o Easy application tuning at high level
  o Best performance / development time ratio