Manycore Application Migration: Methodology and Tools

Московский государственный университет, Лоран Морен, 29 август 2011
Methodology to Port Applications

Phase 1
- Hotspots:
  - Performance goal
  - Validation process
  - Hotspots selection
  - Continuous integration
  - Hours to Days

- Parallelization:
  - Optimize CPU code
  - Exhibit Parallelism
  - Move kernels to GPU
  - Days to Weeks

Phase 2
- Define your parallel project
- Port your application on GPU
- Optimize your GPGPU application

- Tuning:
  - Exploit CPU and GPU
  - Optimize kernel execution
  - Reduce CPU-GPU data transfers
  - Weeks to Months

GPGPU operational application with known potential
Take a decision

- **Go**
  - Dense hotspot
  - Fast GPU kernels
  - Low CPU-GPU data transfers

- **Midterm perspective:**
  Prepare to manycore parallelism

- **No Go**
  - Flat profile
  - GPU results not accurate enough
    - cannot validate the computation
  - Slow GPU kernels
    - i.e. no speedup to be expected
  - Complex data structures
  - Big memory space requirement
Validate your Computation

• Parallel execution on Many-core architectures will (probably) change Floating-Point computation results…
  o Parallel programming change in an extensive way the computation order
  o Hardware floating-point units are not always IEEE 754 compliant
  o Convergence speed of Iterative algorithms can be impacted
  o …

• … but it does not imply that the computation is wrong!
  o A floating-point computation is always an approximation

• The validation must be provided by application providers
  o It is the mathematicians or physicists responsibility
  o They may or may not require binary equal results
  o If not, a validation protocol must be provided
Tools to improve the methodology productivity
Tools in the Methodology

- **Hotspots**
  - Profiling tools: 
    - Gprof, KacheGrind, Vampir, Acumem, ThreadSpotter
  - Debugging tools: gdb, alinea DDT, TotalView

- **Parallelization**
  - HMPP Workbench
  - HMPP Wizard

- **Define your parallel project**
  - Hours to Days

- **Port your application on GPU**
  - Days to Weeks

- **Optimize your GPGPU application**
  - Weeks to Months

- **Phase 1**
  - GPGPU operational application with known potential

- **Phase 2**
  - DEFINE YOUR PARALLEL PROJECT
  - PORT YOUR APPLICATION ON GPU
  - OPTIMIZE YOUR GPGPU APPLICATION
Profiling
Focus on Hotspots (Gprof example)

- Select Hotspots
- Estimate speedup using Amdahl’s law:

\[
\frac{1}{(1 - P) + \frac{P}{S}}
\]
Move Computation to GPU: HMPP Workbench
What is HMPP? (Hybrid Manycore Parallel Programming)

- A directive based programming model
  - Provide an incremental tool to exploit GPU in legacy applications
  - Complement existing parallel APIs (OpenMP or MPI)
  - Avoid exit cost, future-proof solution
  - Based on an OpenStandard

- An integrated compilation chain
  - Code generators from C and Fortran to GPU (CUDA or OpenCL)
  - Source to source compiler: uses hardware vendor SDK
  - One single compiler driver interfacing GPU compilers
  - Complements existing parallel APIs (OpenMP or MPI)

- A runtime support of heterogeneous architectures
  - Manages hardware resource and its availability
  - Interface transfers and synchronizations of the hardware
  - Fast and easier application deployment on new hardware
  - Free of use
HMPP usage in three steps

1. A set of directives to program hardware accelerators
   • Drive your HWAs, manage transfers
   • Optimize kernels at source level

2. A complete tool-chain to build manycore applications
   • Build your hybrid application

3. A runtime to adapt to platform configuration
HMPP Directives: drive Hybrid Applications

```c
int main(int argc, char **argv) {
    // ...
    #pragma hmpp compute advancedload, args[M;N]
    for (j = 0; j < 2; j++) {
        #pragma hmpp compute callsite
compute(size, size, size, alpha, vin1, vin2, beta, vout);
    }
    // ...
}
```

```c
#pragma hmpp compute codelet, target=CUDA, args[a].io=inout
void compute( int M, int N, float alpha, float a[n][n], float b[n][n]) {
    #pragma hmppcg hmppcg grid blocksize 16 X 16
    #pragma hmppcg unroll 2, jam
    for (i = 0; i < M; i += 1) {
        #pragma hmpp unroll 4, guarded
        for (j = 0; j < N; j += 1) {
            a[i][j] = cos(a[i][j]) + cos(b[i][j]) - alpha;
        }
    }
}
```
#pragma hmpp sgemm codelet, target=CUDA, args[vout].io=inout
textern 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 );
  }
  // ...
}
HMPP BLAS Performance on NVIDIA Fermi

SGEMM Performance

2 x Intel(R) Xeon(R) X5560 @ 2.80GHz (8 cores) - MKL
NVidia Tesla C2050, ECC activated – HMPP, CUBLAS, MAGMA
Build a Parallel Project: HMPP Wizard
Make your kernels more GPU friendly
HMPP Wizard

A diagnosis tool to make the code “GPU Friendly”
• Features:
  o Generate GPU porting advices of C/Fortran source code
    • Unsupported programming (pointers, goto-labels, …)
    • Loop structure and parallelism validation
    • Diagnosis the loop nest translation process into a GPU grid of threads
  o Estimate kernels behavior on a GPU and provide specific advices
    • Reduction inside a kernel;
    • Bad memory coalescing;
    • Inefficient memory coalescing;
    • Too low computation density;
    • Detection of well-known patterns such as convolutions
  o Propose basic GPU Tuning directives
• Core technology: based on static analysis
  o Analyze memory access patterns on the GPU
  o Takes into account the full HMPP code generation process
    • Evaluates HMPPCG optimization directives
Analyze your memory access pattern

Use HMPPCG Directives and make your kernel GPU friendly
Tune your application with the HMPP Wizard

An advice: is the loop GPU friendly?

- Example: memory access pattern over the Z dimension (3D)
  - The GPU grid is 2D → the coalescing must be on the X dimension
- A pragma permute is proposed to fix the problem

```c
#pragma hmppcg permute(k,i,j)
for (k = 0 ; k < zMax ; k++)
  for (j = 0 ; j < yMax ; j++)
    for (i = 0 ; i < xMax ; i++)
      {
      float xTrace = i - dto * velx[k][j][i];
      float yTrace = j - dto * vely[k][j][i];
      float zTrace = k - dto * velz[k][j][i];
      xTrace = fminf(fmaxf((xTrace), (1.5f)), (xMax - 2.5f));
      yTrace = fminf(fmaxf((yTrace), (1.5f)), (yMax - 2.5f));
      zTrace = fminf(fmaxf((zTrace), (1.5f)), (zMax - 2.5f));
      int x0 = (int) xTrace;
      }
Debug your hybrid & parallel application: Alinea DDT
• Software tools company since 2001
  – Allinea DDT – the scalable parallel debugger
  – Allinea OPT – the optimization tool for MPI and non-MPI

  – Users at all scales – at 1 to 100,000 cores and above
  – World's only Petascale debugger!

_Simplifying the challenge of multi- and many-core development_

  – Bugs at _scale_ need a debugger at _scale_
    • … until recently debuggers limited to ~4,000-8,000 cores
  – Bugs on _GPUs_ need a debugger for _GPUs_
    • … until recently GPU software couldn't be debugged
Debuggable inside C kernels (codelets) on the GPU

F90 multi-dimensional arrays support in progress for the GPU

Auto-reporting of CAPS runtime errors to the DDT GUI

Also able to debug codelets running on the CPU
Optimize CPU-GPU integration: PARAVER
Analyze the CPU-GPU Transfers
HMPP + PARAVER

Get a precise view of HMPP element behavior
Transfer time, execution time…
Optimize GPU Performance: HMPP Performance Analyzer
Understand how well kernels are performing
HMPP Performance Analyzer

Provide information about the GPU behavior at source level for HMPP applications

• Main features
  o Analyze execution profiles made on the GPU by target specific profilers
  o Compute synthetic metrics from raw profile data
    • Memory throughput, grid size, computation density…
  o Provide detailed statistics from raw profile data

• Benefits
  o Avoid using target specific profiling tools for HMPP applications
    • Nvidia insight profiles Cuda code !
  o Get precise and specific information about the loop nest behavior
  o Explore and exploit at best the GPU power from the source level
  o Fine tune kernels using HMPPCG directives
HMPP Performance Analyzer: visual

Get precise and specific information about the kernel behavior.

Explore and Exploit at best the GPU power from the C source level.

29 августа 2011
Москва - Лоран Морен
HMPP Performance Analyzer: example

Global view of the execution time distribution

Details statistics of raw profiling data

Synthetic kernel metrics

Codelet

```
REAL(realsize) prod, vla, v2a

DO j=1, p
   DO i=1, m
      DO k=1, n
         vout(i, j) = vout(i, j) + vin1(i, k) * alpha
      END DO
   END DO
END DO
END DO

DO j=1, p
   DO i=1, m
      prod = 0
      k = 1
      prod = prod + vout(i, j) = a
      vla = vin1(i, k)
      v2a = vin2(k)
      prod = prod
      END DO
   END DO

END SUBROUTINE compute_hmpp
```
DevDeck
The fastest way to GPGPU porting
An ideal package for a proven methodology

DevDeck

- A multi-level tool suite for manycore applications definition, porting and optimization
- A GPU focused porting methodology and a state-of-the-art products ecosystem
- A resources tank to guide you in your porting issues
DevDeck PACKAGES

DevDeck DEVELOPER

For developers’ workstations

- HMPP HYBRID COMPILER
- DEVELOPMENT TOOLS
  - HMPP Wizard
  - HMPP Performance Analyzer
  - ALLINEA DDT
- SCIENTIFIC LIBRARIES

Node-Locked license

DevDeck ENTERPRISE

For computing centers & large companies

- HMPP HYBRID COMPILER
- DEVELOPMENT TOOLS
  - HMPP Wizard
  - HMPP Performance Analyzer
- CUSTOMIZED SERVICES
  - expertise, tools recommendation, training…

Floating license

DevDeck OEM

For constructors / Integrators

- HMPP HYBRID COMPILER
- DEVELOPMENT TOOLS
  - HMPP Wizard
  - HMPP Performance Analyzer

OEM license

+ Access to Web Resources according to the package:
HMPP Workbench - Wizard - Performance Analyzer - Allinea DDT - Libraries - Methodology supports - Cookbook - Tutorials - Use cases…
Conclusion
### Cost-effective migration requirements

<table>
<thead>
<tr>
<th>Category</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Go / NoGo Analysis</td>
<td>• Application Diagnosis by CAPS</td>
</tr>
<tr>
<td>Incremental approach</td>
<td>• Provided by HMPP</td>
</tr>
<tr>
<td>Don’t redevelop what already exists</td>
<td>• Use GPU libraries/function into HMPP</td>
</tr>
<tr>
<td>Understand finely the data transfers</td>
<td>• Use tools like paraver, Vampire, TAU</td>
</tr>
<tr>
<td>Good understanding of the HW</td>
<td>• HMPP Performance Analyser</td>
</tr>
<tr>
<td>Lots of experimentation at a lower cost</td>
<td>• HMPPCG directives</td>
</tr>
</tbody>
</table>
Gravimetry application

Finite volumes method

- Effort
  - 1 man-month

- Nehalem X5560 improvement:
  - x5.15

- Main porting operation
  - Reduce kernel memory accesses, use of GPU shared memory and optimize Data Movement

The finite volumes method (left) is more accurate than the analytic solution (right) which over estimates the central peak.
Heat transfer simulation

Monte Carlo simulation for thermal radiation

- **Effort**
  - 1.5 man-month

- **Size**
  - ~1kLoC of C d (DP)

- **GPU C2050 improvement**
  - x6 over 8 Nehalem cores

- **Full hybrid version**
  - x23 with 8 nodes over 8 Nehalem cores

- **Main porting operation**
  - adding a few HMPP directives

- **Main difficulty**
  - none
Numerical Weather Prediction

A global cloud resolving model

- **Effort**
  - 1 man-month (part of the code already ported)

- **GPU C1060 improvement**
  - 11x over a Nehalem core
  - 19x over a Harpertown core

- **Main porting operation**
  - Reduction of CPU-GPU transfers

- **Main difficulty**
  - GPU memory size is the limiting factor

29 август 2011
MultiView Stereo

- Resource spent
  - 1 man-month

- Size
  - ~1kLoC of C99 (DP)

- CPU Improvement
  - x 4.86

- GPU C2050 improvement
  - x 120 over serial code on Nehalem

- Main porting operation
  - Rethinking algorithm
Edge detection algorithm

- Sobel Filter benchmark

- Size
  - ~ 200 lines of C code

- GPU C2070 improvement
  - x 25.8 over serial code on Nehalem

- Main porting operation
  - Use of basic HMPP directives
Biosciences, phylogenetics

Phylip, DNA distance

- In association with the HMPP Center Of Excellence for APAC
- Computes a matrix of distances between DNA distances
- Resource spent
  - A first CUDA version developed by Shanghai Jiao Tong University, HPC Lab
  - 1 man-month
- Size
  - 8700 lines of C code, one main kernel (99% of the execution time)
- GPU C2070 improvement
  - x 44 over serial code on Nehalem
- Main porting operation
  - Kernel parallelism & data transfer coalescing leverage
  - Conversion from double precision to simple precision computation