The RapidMind Platform for Portable Programming of Multi-Core Processors and Many-Core Accelerators

Michael McCool
Chief Scientist, RapidMind
SHARCnet Symposium
27 May 2008
RapidMind
Company Overview

- Dedicated to addressing the multicore/manycore challenge
  - Over $10M in funding
- Based on years of research at University of Waterloo
  - One of the first research labs to attempt GP-GPU in 1999
  - Company founded in 2004 based on university GPU prototype
  - Extended to Cell in 2006, x86 CPUs in 2007
  - Widely used, professionally supported software development platform
**Performance**
- Fully leverage the potential of all processors/cores
- Support for adaptive programs and auto-tuning

**Portability**
- Hardware independent applications
- Automatic scalability to additional cores
- Exploit future many-core processors efficiently
- Use accelerators when available

**Programmability**
- Simple extensions of existing practice
- Automate trivia, provide drill-down when needed
- Explicit control of important policy decisions
RapidMind and a GPU accelerator can deliver orders of magnitude performance improvement

GPU Performance

- Financial benchmark performed in partnership with HP
- RapidMind+GPU vs HP’s highly tuned CPU implementation
- **32x to 141x performance advantage with RapidMind**
RapidMind delivers dramatic gains on multi-core CPUs from Intel® and AMD®.

**Multi-core CPU Performance**

- Same HP financial benchmark
- RapidMind + Quad-core CPU vs HP’s highly tuned CPU implementation
- **8x performance advantage with RapidMind on 4 cores**
Porting to the Cell processor with RapidMind provides increased performance with an order of magnitude less work.

**Cell Performance**
- IBM’s benchmark application
- RapidMind and Cell SDK
  - vs
  - Best known implementation on Cell SDK alone
- **RapidMind nearly doubled performance with significantly less developer effort**
Portability
Present and Future Benefits

Deployment choice now, but also future proof
Database transactions
Image/video manipulation
Enterprise search
Data mining
Real-time data analysis
3D visualization
Crowd simulation
Convolution
Electromagnetic simulation
Seismic analysis
Ray tracing (w/ RTT)
Fast Fourier transform
Monte Carlo option pricing
Binomial tree option pricing
Linear algebra
Transformation and lighting
Color and gamma correction
Anisotropic diffusion filtering
Fluid flow
Motion estimation
Collision detection
Rigid body simulation

Broadcast-quality encoding
Medical imaging
Film and television content generation
Image and signal processing
Financial analysis
Seismic analysis
Object tracking
Sorting
Quaternion Julia set
Deferred shading/relighting
Vector (SVG, PDF) textures
Advanced reflectance models
K nearest neighbour search
Keyword search
N-body force computation
Differential equation solver
Optimization under constraints
Tomographic reconstruction
Volume rendering
Time series search
EM time series forecasting
Pattern recognition

Many others...
Objective: Ray tracing of complex automotive and consumer 3D models in real-time

About RTT AG
- A leading global provider of visualization in the automotive, aerospace and consumer goods sector
- Their real-time ray tracer, RealTrace, part of DeltaGen, is software for complex 3D visualization.

Challenges
- Complex images are extremely computationally intensive
- Rendering is typically performed offline, and on multiple servers (called render farms)

RapidMind Solution
- Rendering is now performed real-time
- Smaller projects can be rendered on a single machine with a GPU, or a single Cell BE system

"The RapidMind platform abstracts away the complexities of multi-core programming, allowing RTT to harness the tremendous performance of GPUs. With RapidMind, we can quickly roll out applications that provide our customers with unprecedented capabilities in lightning-fast timelines once thought impossible. Our real-time, interactive RTT RealTrace system is one such example."

-Peter Stevenson, chief operating officer of RTT USA
Objective:
Qbase customers are looking for more **immediate results** for **faster decision support**.

Results:
**Achieved a 17x increase on Intel dual Quad core (8 cores)**
Allows more immediate decision making and more sensors.

- The RapidMind version is **17x faster on a 8 core CPU and is 30x faster on a GPU**
- The same RapidMind implementation can take advantage of single, dual or quad core processors from AMD or Intel and acceleration from ATI or NVIDIA GPUs
Digital Media Test Case

Behavioral Simulation

Cell Blade, PlayStation®3, NVIDIA G80 GPU

Objective: Simulate large number of autonomous characters for film and games

Challenges:
- Simulating emergent social behaviors (flocking, collision avoidance, and distributed decision-making)
- Demands high computation performance
- Not a trivial problem to parallelize: requires inter-core communication

RapidMind Solution
- Simultaneously simulate 16,000 characters on a single Cell processor
- Fully leverages all cores (1-PPE and 8-SPEs) in the Cell BE
- Simulation application built in a single weekend C++ with the RapidMind platform

Developing a simulation of many thousand characters, each with a mind of its own, is a complex task. It requires a large amount of computing power and normally involves a serious effort from the developer in order to achieve reasonable performance.

Utilizing the RapidMind multi-core software platform and a Mercury Cell-based hardware platform, achieving excellent performance of this simulation is made simple.
Digital Media Test Case

Tomographic Reconstruction

AMD® ATI Radeon HD 2900 GPU

Objective: 2D/3D reconstruction of X-ray data in real-time

Challenges
- The reconstruction of cross-sectional images or volumes from Computed Tomography (CT) data into a 2D/3D image
- Typically requires calculating a solution to a very large, sparse linear system of equations
- Data sets are huge, requiring enormous computation performance

RapidMind Solution
- Runs in real-time on an off the shelf computer with a GPU
- The RapidMind solution allows the use an algebraic reconstruction, rather than more complex and computationally expensive algorithms such as Filtered Back Projection or Sparse Matrix
Customer Case Study

Quasi-static Elastography

Objective: Compare productivity and performance of various GPU-based elastography signal processing

Dartmouth Medical School research project

- Implement a real-time elastography signal processing algorithm using the RapidMind platform to increase the performance of GPU-based signal processing

Challenges

- Reducing the image processing time (<1.0 sec) to a point where the results can be used real-time in specific clinical applications such as vascular surgery, and diagnosis of breast or prostate cancer

RapidMind Solution

- RapidMind-enabled algorithm is 100x faster than the non-RapidMind version
- RapidMind-enabled reduces the image processing time from 4 minutes down to one second, without any degradation in image quality

Elastographic imaging approaches depicts tissue elasticity by producing images ultrasonically measured internal tissue strains. Quasi-static Elastography compares the spatial variation of two images taken from slightly different locations, and a cross-correlation analysis is performed on the two images.

The resulting data is invaluable in clinical applications during minimally invasive therapeutic techniques, or as a diagnostic tool.

This type of analysis is computationally expensive and generally difficult to implement on typical computer hardware.
The RapidMind Platform
Overview

• Software development platform for multi-core and many-core processors
• Single-source solution for portable high-performance parallel programming
• Supports high productivity development
• Safe and deterministic general-purpose programming model (SPMD stream)
• Scalable to an arbitrary number of cores
• Can be used to target both accelerators and multi-core processors
• Integrates with existing C++ compilers
A good programming technology should:

1. Provide an accurate conceptual model of the hardware
2. Clearly expose the most important policy decisions and architectural elements of the hardware
3. Provide structure and modularity
4. Automate what can be automated, and not overload the programmer with trivia
5. Provide drill-down mechanisms for use when necessary
1. Parallelism
2. Memory Locality

- Want to expose these at highest level of programming model
- Rest is details, let automation handle
1. **Parallelism**
   - Choose or design a good *parallel* algorithm
   - Must scale with additional processing elements
   - *Needs to be independent of number of cores*

2. **Memory Locality**
   - Efficient use of the memory hierarchy
     - *More frequent use of faster local memory*
   - Planned, coherent use of memory and data transfer
   - High arithmetic intensity
   - *Clear expression of locality*
   - *Clear expression of data movement*
Apply functions to arrays:
• Application: \( C = f(A,B) \)
• May have control flow (SPMD model)
• May have local arrays
• May call other functions
• May access other arrays
• Can read and write to subarrays

Use collective operations:
• Reduce: \( a = \text{reduce}(f,A) \)
• Gather: \( A = B[U] \)
• Scatter: \( A[U] = B \)
• Others . . .
Advantages of SPMD Stream Processing

- Efficient on a variety of architectures
  - Shared memory machines
  - Distributed memory machines
  - Vector/stream machines
  - SIMD-within-a-register/multi-core machines
- Predictable memory access patterns
- Scales to any number of cores
- Single conceptual thread of control
  - Simple extension of existing programming practice
  - No explicit synchronization needed
  - No deadlocks or race conditions
  - Debugging simplified
**API**
- Integrates with C++
- Requires no new tools or workflow

**Platform**
- **Code Optimizer** analyzes and optimizes computations to remove overhead
- **Load Balancer** plans and synchronizes work to keep all cores fully utilized
- **Data Manager** reduces data bottlenecks
- **Logging/Diagnostics** detects and reports performance bottlenecks

**Processor Support Modules**
- ATI/AMD and NVIDIA GPUs
- Cell Blade, Cell Accelerator Board, PS3
- AMD and Intel x86 Multi-core CPUs
Library:
• Use of “canned” functions tuned for performance
• Suitable for some problems, but inflexible

New Language or Language Extensions:
• Major training and implementation hurdle
• Build systems, IDEs, adoption, supported platforms, etc. etc.

Platform:
• *Embedded programming interface*
• Use like a library:
  • Include header file, link, use existing compiler, IDEs, build system, etc.
• As expressive as a language
• Allows adaptive, auto-tuned software development
Computation Capture and Transformation Process

- Interface extracts computation expressed in C++ while eliminating overhead
- Code generator creates native machine code
- Runtime tightly couples multiple optimizations and manages execution over multiple cores

Standard C++ using RapidMind interface

C++ source code

Standard C++ Tools

RapidMind Collection

RapidMind Compilation

Platform specific code

Standard executable with embedded RapidMind operations

Massively parallel computation

RapidMind Execution

Streaming execution

Multicore Processor
<table>
<thead>
<tr>
<th>Purpose</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container for fixed-length data</td>
<td>Value</td>
</tr>
<tr>
<td>Container for variable-sized multidimensional data</td>
<td>Array</td>
</tr>
<tr>
<td>Container for computations</td>
<td>Program</td>
</tr>
</tbody>
</table>
Values

1 half
2 double
Value<3, float>
4 int

Tuple size
Element type
Values

Tuple size

Element type

1h
2d
Value 3f
4i
Arrays

1. Value4d

\[ \text{Array}^{<2, \text{Value3f}>} \]

2. Value2i

Dimensionality

Item type
Program p;

p = BEGIN {
   In<Value3f> a, b;
   Out<Value3f> c;

   Value3f d = f(a, b);
   c = d + a * 2.0f;
} END;
SIMD:

- *Single Instruction, Multiple Data*
- Kernels include sequences of simple instructions
- Take constant amount of time to execute

SPMD:

- *Single Program, Multiple Data*
- Kernels may include control flow (loops and conditionals)
- Can avoid unnecessary work

**SPMD includes but is *intrinsically* more powerful than SIMD**
Program p;

p = BEGIN {
   In<Value3f> a, b;
   Out<Value3f> c;

   Value3f d = f(a, b);
   IF (all(a > 0.0f)) {
      c = d + a * 2.0f;
   } ELSE {
      c = d - a * 2.0f;
   } ENDIF;
} END;
• Apply programs to arrays, get new arrays

\[ C = p(A, B) ; \]

*Invokes parallel execution*
• Apply programs to arrays, get new arrays
  \[ C = p(A,B); \]

• Can use accessors to get subarrays
  \[ \text{slice}(C,500,999) = p(\text{slice}(A,0,499),\text{take}(B,500)); \]

• Can use programs as parameters to collectives
  \[ a = \text{reduce}(p,B); \]
Usage:
- Include platform header
- Link to runtime library

Data:
- Tuples
- Arrays
- Global data abstraction

Programs:
- Defined dynamically
- Execute on coprocessors
- Remote procedure abstraction

```cpp
#include <rapidmind/platform.hpp>
using namespace rapidmind;

Value1f f = 2.0f;
Array<2,Value3f> a(512,512);
Array<2,Value3f> b(512,512);

Program prog = BEGIN {
  In<Value3f> r, s;
  Out<Value3f> q;
  q = (r + s) * f;
} END;

a = prog(a,b);
f = 3.0f;
stride(a,2,2) = prog(
  slice(a,0,255,0,255),
  slice(b,256,511,0,255));
```
Computational efficiency:
- Permit user to directly express **parallelism**
- **Start** with a large amount of parallelism
- Hierarchically subdivide over **multiple** levels of hardware parallelism mechanisms
- **Select granularity suitable for each level**
- Compile out overhead with staged code generation

Data movement efficiency:
- Permit user to directly express **data locality**
- Eliminate unnecessary data movement
- Align and organize data to permit vectorization and avoid unnecessary interprocessor chatter
- Transform data structures into more efficient implementations
- Automate latency hiding (multibuffering, etc)
Multiple parallelism mechanisms available:
• Instruction pipelining
• Superscalar instruction issue
• SIMD within a register vectorization (SWAR)
• Overlapping memory access with computation
• Multithreading on one core
• Multi-core workload distribution
• Multi-processor workload distribution
• Asynchronous host and “co-processor” execution

RapidMind’s Top-down approach:
• User specifies a large amount of parallelism
• Platform hierarchically subdivides
Exploiting Multiple Levels of Parallelism

User specifies:

System implements:

Example uses:
- Two cores
- Four-way vectorization
- Latency hidden with streaming

*Actual distribution of work depends on hardware*
Memory bottleneck:

- Cores share finite off-chip bandwidth
- Latency cannot be improved significantly

⇒ and is already 100’s of cycles to access DRAM
• To hide latency, need **more** parallelism than number of cores, *schedule* data:

\[
\text{Concurrency} = \text{parallelism} \times \text{latency}
\]
RapidMind on Accelerators
Remote Memory Management

Without data management

With data management
Multiple programs can be fused together, further reducing data movement.

With program fusion,
Multicore is a major disruption

• All computers will be massively parallel
• All programmers will have to write parallel programs
• **Software development challenge**

Programming models are important!

• Data parallelism has scalability and safety advantages
• SPMD Stream Programming model
• Intrinsically safe, drill-downs only when needed

Programming platforms:

• *Not* necessary to introduce completely new languages
• Can obtain similar performance and expressiveness within *standard* C++ and *existing* compilers