8x Performance on Multi-core CPU with RapidMind

RapidMind demonstrates how their platform gives developers an 8x performance advantage on an Intel® Core2 Quad CPU.

RapidMind Inc. performed a preliminary benchmark on a prototype implementation of the RapidMind platform for x86-based CPUs. The current RapidMind platform provides software developers with a simple way to achieve great performance on the GPU and Cell BE. This benchmark was originally developed in partnership with Hewlett-Packard to evaluate GPU performance. The new results demonstrate how the RapidMind platform will provide similar benefits to developers targeting multi-core Intel and AMD® CPUs.

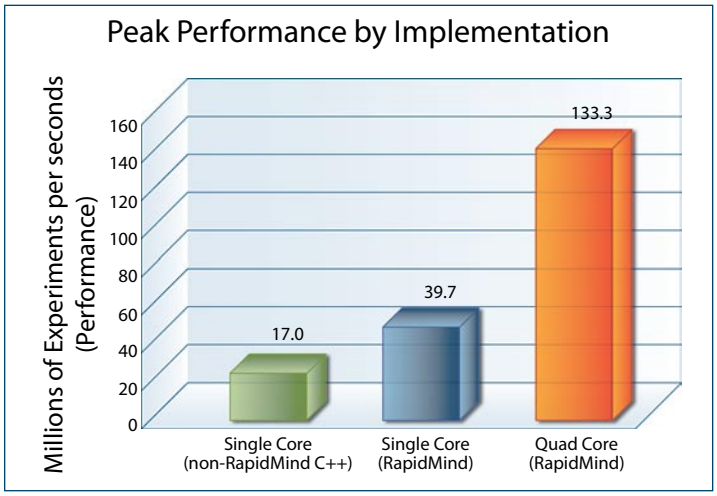
Results from a non-RapidMind, yet optimized implementation, were compared to a RapidMind-enabled version. The results demonstrate that the RapidMind-enabled implementation:

- Is as simple as creating a typical single-threaded implementation for a single core
- Doubles the performance of the highly optimized, non-RapidMind version on a single core
- Automatically scales to multiple cores (in this case 4) and achieves 8 times the performance of the single-core, non-RapidMind version

The Application Developer's Challenge

The existing programming ecosystem is not equipped to tap the enormous performance benefits of multi-core processors. Traditional serial programs written using a typical C++ programming approach will not perform well on multi-core processors. Programming multi-core processors is also challenging, as task-based thread programming methodologies introduce issues of synchronization, deadlock, load balancing and non-determinism. Additionally, multi-core architectures put pressure on the memory system, as the gap between off-chip bandwidth and on-chip processing power continues to grow exponentially.

While traditional development tools are insufficient, the RapidMind platform is designed to address these gaps. With RapidMind, software developers are able to program in standard C++, using their existing compilers and IDEs, and the resulting applications are portable and currently run on either GPUs or the Cell BE.

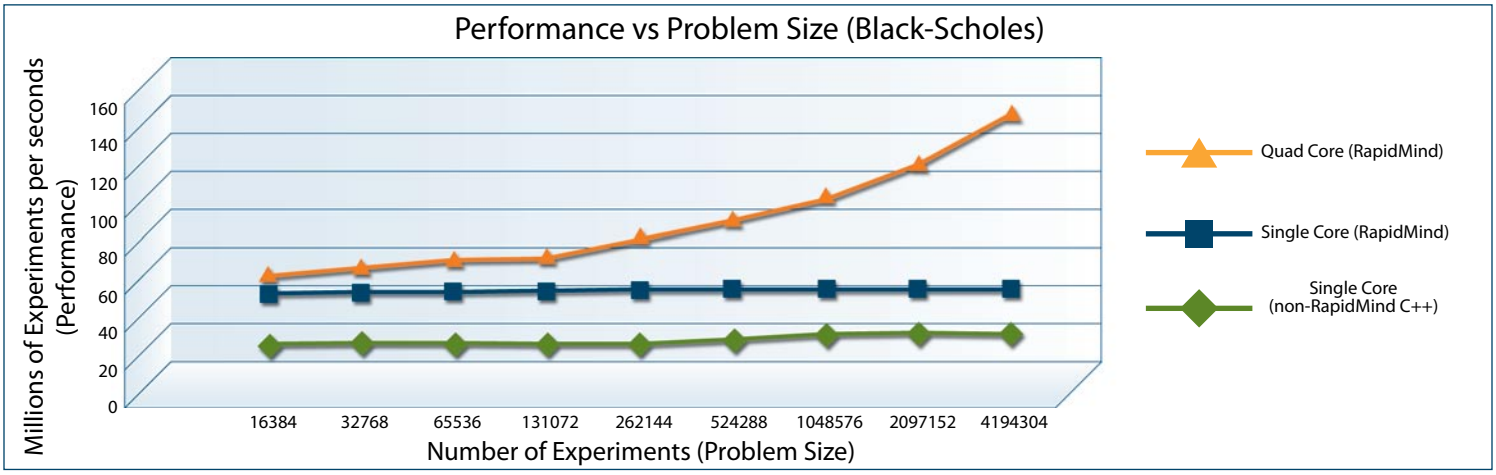


Increased Performance with Less Effort

This chart shows the comparative performance gain using the RapidMind platform relative to a non-RapidMind implementation. The non-RapidMind implementation was optimized by scientists from Hewlett-Packard in a previous study to effectively exploit SSE and leverage the highest compiler optimization and auto-vectorization settings in Intel's ICC optimizing compiler.

The non-RapidMind implementation achieved the results expected from a single-threaded application on a single core. The RapidMind-enabled application was as easy to implement, but automatically takes advantage of all four available cores, and out performed the single core implementation by a factor of 8 times!

The following chart shows the same calculation performed on different problem sizes, demonstrating that the performance benefit increases with larger and more arithmetically intense problems.



Comparing Implementations

A Black-Scholes benchmark was implemented that evaluates European option prices via quasi-Monte Carlo sampling. Discussion of the actual benchmark used can be found in a previous joint paper by scientists from RapidMind and Hewlett-Packard Company, ***Performance Evaluation of GPUs using the RapidMind Development Platform*** (http://www.rapidmind.net/sc06_hp_rapidmind_cpugpu_summary.php).

Typical C++ code

```
for (int i = 0; i < num_experiments; i++) {
   float i1 = float(i + 0.5f) / num_experiments;
   float i2 = bitreverse(i + 1);

   boxmuller_shirley(i1, i2, phi_const[0],
   phi_const[1]);

   s1_const[0] = S_0 * expf(R + SDT * phi_const[0]);
   s1_const[1] = S_0 * expf(R + SDT * phi_const[1]);

   CT = std::max(s1_const[0] - K, 0.0f);
   sumCT = sumCT + CT;
   sumCT2 = sumCT2 + CT*CT;

   CT = std::max(s1_const[1] - K, 0.0f);
   sumCT = sumCT + CT;
   sumCT2 = sumCT2 + CT*CT;

}
```

RapidMind-enabled C++ code

```
Program blackscholes = BEGIN {
   In<Value1i> i;

   Value2f hammersley = join((i + 0.5f) /
   num_experiments, bitreverse (i + 1));

   Value2f phi = boxmuller_shirley(hammersley);

   Value2f S = S_0 * exp(R + SDT * phi);
   Value2f CT = rapidmind::max(0.0f, S - K);

   Out<Value2f> CT_CT2 = join(sum(CT), dot(CT, CT));
} END;

// ...

Value2f sum_of_CT_and_CT2 = sum<Value2f>(blackscholes
(grid (num_experiments)));
```

The right-hand figure shows the main portion of the RapidMind-enabled C++ code used to implement the algorithm. The left-hand figure shows the non-RapidMind single-threaded C++ code used for the performance comparisons. Note that the non-RapidMind implementation included additional hand optimizations which were not included here for brevity, whereas the RapidMind-enabled code is the actual code used to generate the performance numbers. By comparing the two implementations, it is simple to see how they correspond. However, the RapidMind implementation made use of the fact that RapidMind values can contain more than a single scalar and perform operations across them, leading to shorter code to express the same computation.

While the non-RapidMind code iterates through the experiments serially to accumulate the final result, the RapidMind-enabled code expresses this parallel operation explicitly. This explicit statement of parallelism allows the RapidMind platform to generate efficient code for each core using techniques such as vectorization, and distribute the problem across an arbitrary number of cores automatically and efficiently. With RapidMind, the developer continues to use C++ and their existing compilers and IDEs. However, the RapidMind platform completely eliminates the overhead of C++, so the modularity constructs of C++ can still be used to structure large applications without sacrificing performance.

Conclusions

This benchmark clearly demonstrates that the RapidMind platform makes multi-core programming as easy as single-threaded, single core programming, yet takes full advantage of all available cores. The RapidMind C++ implementation also significantly outperforms non-RapidMind C++ implementations on multi-core or single core processors. The current RapidMind platform supports the GPU and the Cell BE processors and will support x86 multi-core processors later in 2007. With RapidMind, developers can build their applications today knowing that not only will they perform well on the GPU and the Cell BE, but also that they will perform well on multi-core CPUs in the future.

About RapidMind Inc.

RapidMind provides a software development platform that allows software vendors to deliver high performance on multi-core and stream processors, including the GPU and the Cell BE. Without sacrificing development simplicity, RapidMind-enabled applications experience a dramatic leap in performance.

470 Weber St N • Unit 203 T +1 519.885.5455 info@rapidmind.net
Waterloo, Ontario • N2L 6J2 F +1 519.885.1463 www.rapidmind.net

RAPIDMIND