Accelerated Computation of the Voigt Function

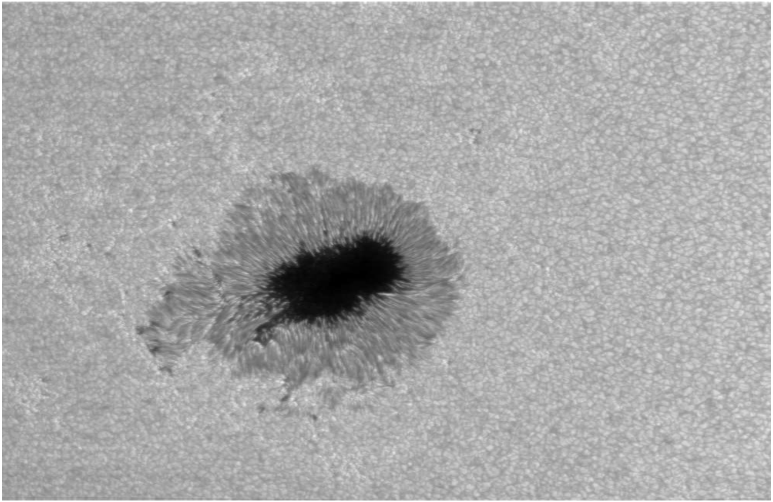
Experiences on the Cell BE and NVIDIA GPUs

Jose Garcia, Rory Kelly

Computational & Information System Laboratory, NCAR, Boulder, CO

December 15, 2008

HINODE Data: Stokes I



The Voigt Function: What is it?

Gaussian Distribution

- Doppler broadening due to the thermal velocities in the medium. This is described by a Normal profile with probability function given by a *Gaussian* distribution.

The Voigt Function: What is it?

Gaussian Distribution

- Doppler broadening due to the thermal velocities in the medium. This is described by a Normal profile with probability function given by a *Gaussian* distribution.

Gaussian Distribution

$$G(f) = \frac{1}{\alpha_D\sqrt{\pi}} e^{\frac{-(f-f_0)^2}{\alpha_D^2}} \tag{1}$$

The Voigt Function: What is it?

Cauchy-Lorentz Distribution

- Natural broadening due to the uncertainty principle.

The Voigt Function: What is it?

Cauchy-Lorentz Distribution

- Natural broadening due to the uncertainty principle.
- Resonance broadening.

The Voigt Function: What is it?

Cauchy-Lorentz Distribution

- Natural broadening due to the uncertainty principle.
- Resonance broadening.
- Collision broadening.

The Voigt Function: What is it?

Cauchy-Lorentz Distribution

- Natural broadening due to the uncertainty principle.
- Resonance broadening.
- Collision broadening.

Cauchy-Lorentz Distribution

$$L(f) = \frac{1}{\pi}\frac{\alpha_L}{(f - f_0)^2 + (\alpha_L^2)} \tag{2}$$

The Voigt Function: What is it?

The Voigt Function is the convolution of G and L

$$(L * G)(f) = \int_{-\infty}^{\infty} L(\tau)G(f - \tau)d\tau \; . \tag{3}$$

The Voigt Function: What is it?

The Voigt Function is the convolution of G and L

$$(L * G)(f) = \int_{-\infty}^{\infty} L(\tau) G(f - \tau) d\tau \, . \tag{3}$$

In Analytical Form

$$P(x, y) = \frac{1}{\alpha_D} \left(\frac{ln(2)}{\pi} \right)^{\frac{1}{2}} \frac{y}{\pi} \int_{-\infty}^{\infty} \frac{e^{(-t^2)}}{y^2 + (x - t)^2} dt \tag{4}$$

The Voigt Function: What is it?

The Voigt Function is the convolution of G and L

$$(L * G)(f) = \int_{-\infty}^{\infty} L(\tau)G(f-\tau)d\tau \ . \tag{3}$$

In Analytical Form

$$P(x,y) = \frac{1}{\alpha_D}\left(\frac{ln(2)}{\pi}\right)^{\frac{1}{2}} \frac{y}{\pi} \int_{-\infty}^{\infty} \frac{e^{(-t^2)}}{y^2+(x-t)^2} dt \tag{4}$$

We use the following approximation

$$erfc(z) \approx R(z) = \frac{\sum_{i=0}^{p} a_i z^i}{z^{p+1} + \sum_{i=0}^{p} b_i z^i} \ . \tag{5}$$

See: Pierluissi, J.: Fast Calculational Algorithm for the Voigt Profile. Journal of Quantitative Spectroscopy and Radiative Transfer, 18 (1977)

Test Problem

Voigt function evaluated on a 2D grid

- One function evaluation per grid point.

Test Problem

Voigt function evaluated on a 2D grid

- One function evaluation per grid point.
- Three test case sizes to evaluate scalability:
  - Small (4096^2) points.
  - Medium (8192^2) points.
  - Large (16384^2) points.

Test Problem

Voigt function evaluated on a 2D grid

- One function evaluation per grid point.
- Three test case sizes to evaluate scalability:
  - Small (4096^2) points.
  - Medium (8192^2) points.
  - Large (16384^2) points.

Each function evaluation:

- Uses 32-bit floating point precision.

Test Problem

Voigt function evaluated on a 2D grid

- One function evaluation per grid point.
- Three test case sizes to evaluate scalability:
  - Small (4096^2) points.
  - Medium (8192^2) points.
  - Large (16384^2) points.

Each function evaluation:

- Uses 32-bit floating point precision.
- Requires 2 inputs (8 bytes), produces 2 outputs (8 bytes).

Test Problem

Voigt function evaluated on a 2D grid

- One function evaluation per grid point.
- Three test case sizes to evaluate scalability:
  - Small (4096^2) points.
  - Medium (8192^2) points.
  - Large (16384^2) points.

Each function evaluation:

- Uses 32-bit floating point precision.
- Requires 2 inputs (8 bytes), produces 2 outputs (8 bytes).
- Contains 96 floating point operations (Flops).

Test Problem

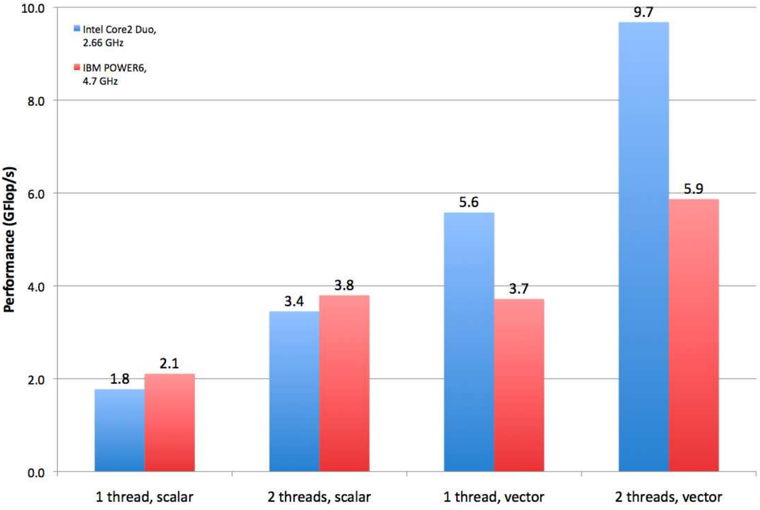
Voigt function evaluated on a 2D grid

- One function evaluation per grid point.
- Three test case sizes to evaluate scalability:
  - Small (4096^2) points.
  - Medium (8192^2) points.
  - Large (16384^2) points.

Each function evaluation:

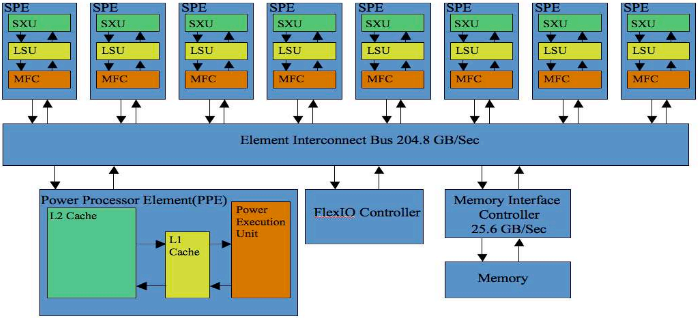
- Uses 32-bit floating point precision.
- Requires 2 inputs (8 bytes), produces 2 outputs (8 bytes).
- Contains 96 floating point operations (Flops).
- Has relatively low computational intensity (6 Flops/byte). Expect memory transfers to be a determiner of performance.

Performance Baseline: Highly Optimized Microprocessor Version

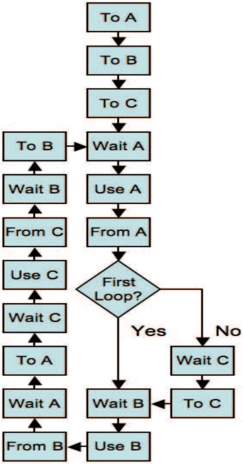


The Cell BE Test System: IBM QS-22

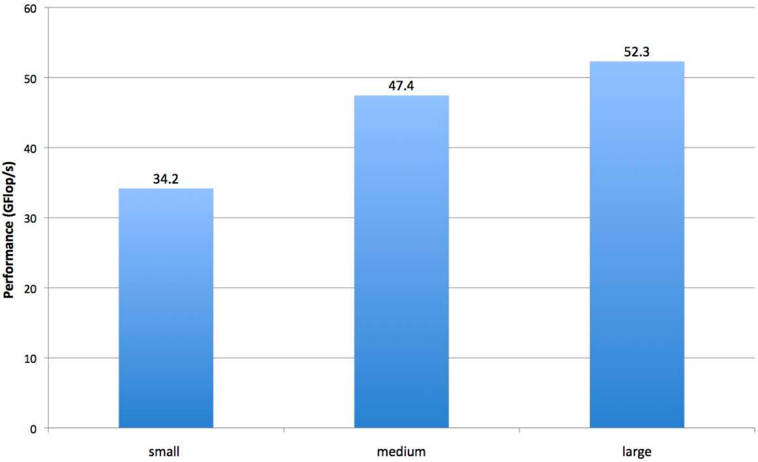
- Contains two PowerXCell 8i Cores, with 16 SPEs.
- 32 GB of globally accessible system memory.
- Peak computational performance of 409.6 GFlop/s.



DMA Multibuffering Scheme



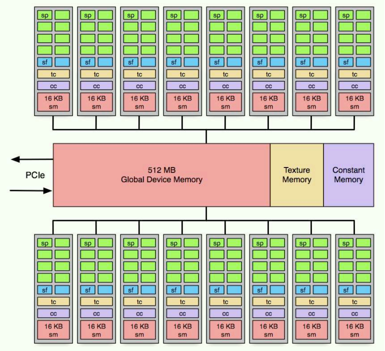
Performance of the IBM QS22 Blade



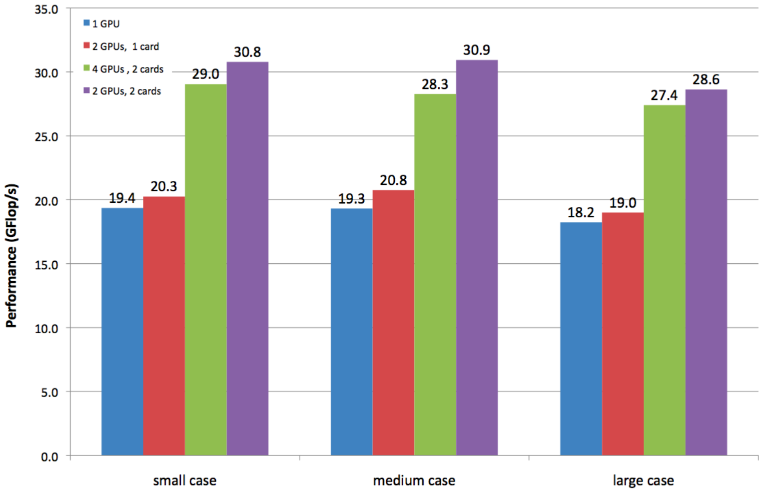
1. Introduction
2. The Cell BE
3. NVIDIA Graphics Card
4. Comparisons & Conclusions

The NVIDIA Test System

- Quad-core Opteron, two NVIDIA GeForce 9800 GX2 cards.
- 4.0 GB of CPU memory, 1.0 GB of GPU memory per card.
- Each GPU card contains two G92 cores, each with 128 SPs.
- Peak computational performance in excess of 1.5 TFlop/s.



Performance of the NVIDIA GeForce 9800 GX2 Card



Performance Considerations

Host to Device Bandwidth

- For single card PCIe 1.0 limits data transfer to 4.0 GB/s

Performance Considerations

Host to Device Bandwidth

- For single card PCIe 1.0 limits data transfer to 4.0 GB/s
- In practice, available PCIe bandwidth is about 3.5 GB/s

Performance Considerations

Host to Device Bandwidth

- For single card PCIe 1.0 limits data transfer to 4.0 GB/s
- In practice, available PCIe bandwidth is about 3.5 GB/s
- Max performance: 6 Flop/byte $\times$ 3.5 GB/s = 21 GFlop/s.

Performance Considerations

Host to Device Bandwidth

- For single card PCIe 1.0 limits data transfer to 4.0 GB/s
- In practice, available PCIe bandwidth is about 3.5 GB/s
- Max performance: 6 Flop/byte $\times$ 3.5 GB/s $=$ 21 GFlop/s.

Memory Bandwidth

- For two cards, CPU memory bandwidth is the constraint.

Performance Considerations

Host to Device Bandwidth

- For single card PCIe 1.0 limits data transfer to 4.0 GB/s
- In practice, available PCIe bandwidth is about 3.5 GB/s
- Max performance: 6 Flop/byte $\times$ 3.5 GB/s $=$ 21 GFlop/s.

Memory Bandwidth

- For two cards, CPU memory bandwidth is the constraint.
- Useable CPU memory bandwidth is about 5.7 GB/s.

Performance Considerations

Host to Device Bandwidth

- For single card PCIe 1.0 limits data transfer to 4.0 GB/s
- In practice, available PCIe bandwidth is about 3.5 GB/s
- Max performance: 6 Flop/byte $\times$ 3.5 GB/s $=$ 21 GFlop/s.

Memory Bandwidth

- For two cards, CPU memory bandwidth is the constraint.
- Useable CPU memory bandwidth is about 5.7 GB/s.
- Max performance: 6 Flop/byte $\times$ 5.7 GB/s $=$ 34.2 GFlop/s.

Performance Considerations

Host to Device Bandwidth

- For single card PCIe 1.0 limits data transfer to 4.0 GB/s
- In practice, available PCIe bandwidth is about 3.5 GB/s
- Max performance: 6 Flop/byte $\times$ 3.5 GB/s $=$ 21 GFlop/s.

Memory Bandwidth

- For two cards, CPU memory bandwidth is the constraint.
- Useable CPU memory bandwidth is about 5.7 GB/s.
- Max performance: 6 Flop/byte $\times$ 5.7 GB/s $=$ 34.2 GFlop/s.

The GPUs are Starving for Data

- Data can be computed much faster than it can be transferred.

Performance

Performance

- Algorithm is dominated by moving data to floating point units.

Performance

- Algorithm is dominated by moving data to floating point units.
- On Cell, bandwidth is not limited by hardware. Could possibly be increased with a deeper multi-buffering scheme.

Performance

- Algorithm is dominated by moving data to floating point units.
- On Cell, bandwidth is not limited by hardware. Could possibly be increased with a deeper multi-buffering scheme.
- On GPU, bandwidth could be increased with faster memory and PCIe 2.0, but only by 2x in the best case.

Performance

- Algorithm is dominated by moving data to floating point units.
- On Cell, bandwidth is not limited by hardware. Could possibly be increased with a deeper multi-buffering scheme.
- On GPU, bandwidth could be increased with faster memory and PCIe 2.0, but only by 2x in the best case.
- Even with only 96 Flops per evaluation both accelerators outperform the highly optimized, hand-tuned, threaded and vectorized microprocessor code by 3x - 5x.

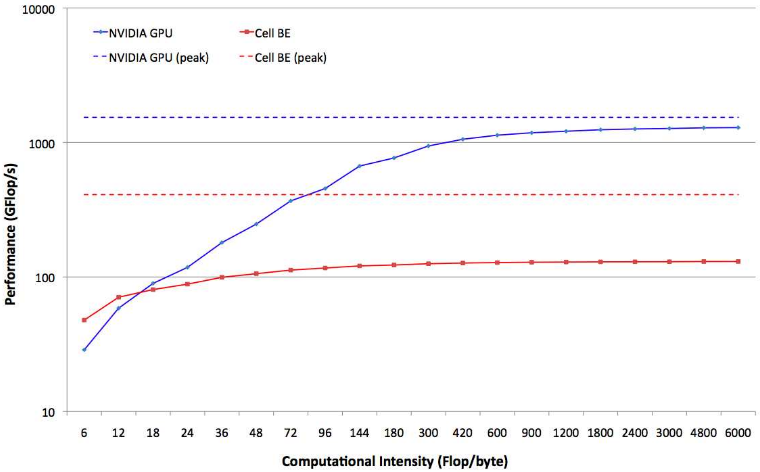
Performance

- Algorithm is dominated by moving data to floating point units.
- On Cell, bandwidth is not limited by hardware. Could possibly be increased with a deeper multi-buffering scheme.
- On GPU, bandwidth could be increased with faster memory and PCIe 2.0, but only by 2x in the best case.
- Even with only 96 Flops per evaluation both accelerators outperform the highly optimized, hand-tuned, threaded and vectorized microprocessor code by 3x - 5x.
- However, achieving a large fraction of peak performance for a problem of low computational intensity is not realistic on either platform.

Performance

- Algorithm is dominated by moving data to floating point units.
- On Cell, bandwidth is not limited by hardware. Could possibly be increased with a deeper multi-buffering scheme.
- On GPU, bandwidth could be increased with faster memory and PCIe 2.0, but only by 2x in the best case.
- Even with only 96 Flops per evaluation both accelerators outperform the highly optimized, hand-tuned, threaded and vectorized microprocessor code by 3x - 5x.
- However, achieving a large fraction of peak performance for a problem of low computational intensity is not realistic on either platform.
- How will performance scale if we artificially increase the computational intensity?

Performance with Increasing Computational Intensity



Software Development and Programmability

Software Development and Programmability

- In our opinion, programming in CUDA is a little simpler than programming with the Cell SDK.

Software Development and Programmability

- In our opinion, programming in CUDA is a little simpler than programming with the Cell SDK.
- In CUDA the main conceptual shift is from accessing data elements by loop index to accessing data elements in parallel with indices mapped to threads. The difficulty is similar to using OpenMP for programming SMPs.

Software Development and Programmability

- In our opinion, programming in CUDA is a little simpler than programming with the Cell SDK.
- In CUDA the main conceptual shift is from accessing data elements by loop index to accessing data elements in parallel with indices mapped to threads. The difficulty is similar to using OpenMP for programming SMPs.
- Programming with the Cell SDK is more difficult, but uses familiar Unix HPC programming concepts, e.g. interprocess communication, Pthreads, vectorization, etc. and the Cell SDK can be learned quickly by a programmer with this background.

Software Development and Programmability

- In our opinion, programming in CUDA is a little simpler than programming with the Cell SDK.
- In CUDA the main conceptual shift is from accessing data elements by loop index to accessing data elements in parallel with indices mapped to threads. The difficulty is similar to using OpenMP for programming SMPs.
- Programming with the Cell SDK is more difficult, but uses familiar Unix HPC programming concepts, e.g. interprocess communication, Pthreads, vectorization, etc. and the Cell SDK can be learned quickly by a programmer with this background.
- Additionally, the Cell SDK may provide a more flexibility in certain applications, and capabilities not available in CUDA.

Future Work

We plan to continue exploring the capabilities of these platforms. In the near term we will be looking at numerical methods and physical simulations with more complicated data dependencies, and more complex computational workloads.

Future Work

We plan to continue exploring the capabilities of these platforms. In the near term we will be looking at numerical methods and physical simulations with more complicated data dependencies, and more complex computational workloads.

Acknowledgements

This research was made possible by generous donations of time, resources and equipment from IBM and NVIDIA.

Future Work

We plan to continue exploring the capabilities of these platforms. In the near term we will be looking at numerical methods and physical simulations with more complicated data dependencies, and more complex computational workloads.

Acknowledgements

This research was made possible by generous donations of time, resources and equipment from IBM and NVIDIA.

For More Information

You can find the complete paper and the source code from this experiment, as well as more information about our on-going research at: http://www.cisl.ucar.edu/css