Porting Black Holes to the GPU

BB, arXiv:1104.3408
related projects: A. Weyhausen, D. Hilditch, BB; M. Ansorg, R. Panosso
much more physics, two black holes: SpEC, CITA (Caltech, Cornell)
A pseudospectral matrix method for time-dependent tensor fields on a spherical shell implemented on a GPU

Outline:
1. GPU Computing
2. Science Overview
3. Numerical Method
4. Implementation
5. Results
GPU Computing
GPU Computing

Theoretical GFLOP/s

Theoretical GB/s

B. Brügmann, University of Jena, SFB/Transregio 7  http://www.tpi.uni-jena.de
GPU Computing

Performance:
• clock speed: GHz
• CPU: Gflop/s
• memory: Gbyte/s, latency
• interconnect: Gbyte/s, latency
• algorithm: instructions/problem
• coding: lines/day
• output: science/day

Pay 100
A C, run 100 times faster!

Pay 2000
A C, work for months, run small jobs 10 times faster!
GPU Computing

Pay 100€, run 100 times faster !!
GPU Computing

Pay 100€, run 100 times faster?

Performance:
- clock speed: GHz
- CPU: Gflop/s
- memory: Gbyte/s, latency
- interconnect: Gbyte/s, latency
- algorithm: instructions/problem
- coding: lines/day
- output: science/day/€
GPU Computing

Pay 100€, run 100 times faster?

Performance:

- clock speed: GHz
- CPU: Gflop/s
- memory: Gbyte/s, latency
- interconnect: Gbyte/s, latency
- algorithm: instructions/problem
- coding: lines/day
- output: science/day/€

Pay 2000€, work for months, run small jobs 10 times faster!

B. Brügmann, University of Jena, SFB/Transregio 7  http://www.tpi.uni-jena.de
Science Overview

Physics:
- black holes, gravitational waves, general relativity
Science Overview

Physics:
- black holes, gravitational waves, general relativity

Mathematics:
- partial differential equations
  \[ \partial_t u^\mu + A_{i\nu}^\mu(u) \partial_i u^\nu = S^\mu(u), \]
- time-dependent, 3d space
- non-linear
- tensor fields, \( \mu = 1, \ldots, 50 \)
Science Overview

Physics:
- black holes, gravitational waves, general relativity

Mathematics:
- partial differential equations
  \[ \partial_t u^\mu + A^{i\mu,\nu}(u) \partial_i u^\nu = S^\mu(u), \]
- time-dependent, 3d space
- non-linear
- tensor fields, \( \mu = 1, \ldots, 50 \)

Our main effort, SFB/GK/DLR, running at LRZ/Jülich/PRACE:
- BAM code: C (Mathematica), MPI, adaptive mesh refinement
  4th- to 8th-order finite differencing on regular grids
- a few 100k lines of code
Science Overview

Physics:
- black holes, gravitational waves, general relativity

Mathematics:
- partial differential equations
  \[ \partial_t u^\mu + A^{i\mu\nu}(u) \partial_i u^\nu = S^\mu(u), \]
- time-dependent, 3d space
- non-linear
- tensor fields, \( \mu = 1, \ldots, 50 \)

Evaluate new technology and new algorithm:
- GPUs
- pseudospectral method
- single sph. sym. black hole, approx. static, but 3d evolutions
- formulation of Lindblom et al 2006
Numerical Method

Basic choices:

- “wave equation” \( \Rightarrow \) explicit method of lines (RK4)
- isolated BHs \( \Rightarrow \) spherical shell
  spherical coordinate grid \( r \in [r_0, r_1], \theta, \phi \)
- coordinates singular \( \Rightarrow \) Cartesian tensor components
  e.g. \( g_{xy}(t, r, \theta, \phi) \)
- no shocks \( \Rightarrow \) pseudospectral method
- pseudospectral, shell \( \Rightarrow \) Chebyshev in \( r \), Fourier/\( Y_{lm} \) in \( \theta, \phi \)
- non-linear, time-dep. \( \Rightarrow \) filter, here: 2d filter \( \theta, \phi \)
Numerical Method

Special choices:

- **CFF-basis instead of CY-basis**
  - double Fourier basis on sphere: $\theta \in [0, 2\pi]$, $\phi \in [0, 2\pi]$
  - requires filter (grid points cluster near poles)
  - for scalar field CFF-basis/Y-filter $\Leftrightarrow$ CY-basis
  - CFF/Y is in principle simpler and more efficient

Numerical Method

new: CFF/Yn for general tensors

- decompose tensors in spin-weighted spherical harmonics $Y^n_{lm}$
  (Newman, Penrose 1966)
  (Kostelec, Maslen, Rockmore, Healy 2000)
  (Wiaux, Jacques, Vielva, Vanderheynst 2006)
- as opposed to CYn; CY/Yn; CFF/Y
  (Novak et al. 2010; Kidder et al. 2005, Lindblom et al. 2006; Tichy 2009)
- extends “generalized discrete Y-transform”
  (Swarztrauber, Spotz 2000, 2003)
Numerical Method

new: CFF/Yn for general tensors

• decompose tensors in spin-weighted spherical harmonics $Y^n_{lm}$
  (Newman, Penrose 1966)
  (Kostelec, Maslen, Rockmore, Healy 2000)
  (Wiaux, Jacques, Vielva, Vandergheynst 2006)
• as opposed to CYn; CY/Yn; CFF/Y
  (Novak et al. 2010; Kidder et al. 2005, Lindblom et al. 2006; Tichy 2009)
• extends “generalized discrete Y-transform”
  (Swarztrauber, Spotz 2000, 2003)

• clustering/time-stepping and Cart. tensor components and non-linear
  $\Rightarrow$ need Yn-filter anyway

two shortcuts: CFF + Cart.-comps $\Rightarrow$ one cleanup: Yn-filter
Black Hole: evolution on x-axis

R10 25x09x18, gtt t=0
gtx t=0
gxx t=0
gtt t=1000
gtx t=1000
gxx t=1000

R10 25x09x18, rhs gtt t=0
rhs gtx t=0
rhs gxx t=0

t=0 to 700

t=200 to 700
Black Hole: convergence, numerical fix-point

![Graph showing convergence and numerical fix-point](image-url)

- Logarithmic scale for $|\text{inf}|$
- Time axis from 0 to 2000
- Various lines representing different time frames:
  - rhs gtt, R10 13x09x18
  - 19x09x18
  - 25x09x18
  - 31x09x18
  - 37x09x18

B. Brügmann, University of Jena, SFB/Transregio 7  http://www.tpi.uni-jena.de
Black Hole: scalar Y-filter

![Graphs showing the behavior of scalar Y-filter for different lambda values.](image-url)
Black Hole: need tensor Yn-filter
Black Hole: tensor Yn-filter to $t = 200,000$

- "long-term" stable compared to typical black hole runs of $\lesssim 5,000$
- residual linear growth
- to do: proper outer boundary (instead of frozen characteristics)
Implementation

- 3d black holes and pseudospectral $\Rightarrow$ small $N$ problem (!)
  $N \lesssim 50$ (never $\gtrsim 500$) for (i) convergence, (ii) physics scales
  use multiple patches rather than larger $N$

- small $N$ $\Rightarrow$ matrix multiplication MM can be faster than FFT
  just as well since Legendre transform by MM for $N \lesssim 300$
  just as well since Yn-transforms not even available in general
Implementation

- 3d black holes and pseudospectral $\Rightarrow$ small $N$ problem (!)
  $N \lesssim 50$ (never $\gtrsim 500$) for (i) convergence, (ii) physics scales
  use multiple patches rather than larger $N$

- small $N$ $\Rightarrow$ matrix multiplication MM can be faster than FFT
  just as well since Legendre transform by MM for $N \lesssim 300$
  just as well since $Y_n$-transforms not even available in general

$\Rightarrow$ implement derivatives and filters by MM
Implementation

- 3d black holes and pseudospectral ⇒ small $N$ problem (!) 
  $N \lesssim 50$ (never $\gtrsim 500$) for (i) convergence, (ii) physics scales 
  use multiple patches rather than larger $N$

- small $N$ ⇒ matrix multiplication MM can be faster than FFT 
  just as well since Legendre transform by MM for $N \lesssim 300$ 
  just as well since $Y_n$-transforms not even available in general

$\Rightarrow$ implement derivatives and filters by MM

- Typical operation: (small $\times$ small) times (small $\times$ LARGE)
  3d spectral derivative: $A_{37\times37} \times B_{37\times50000}$
  2d non-prod. transform: $A_{K\times37} \times B_{37\times50000}$, $K = 1, \ldots, 37$

\[
\begin{pmatrix}
  x & x & x & x \\
  x & x & x & x \\
  x & x & x & x
\end{pmatrix}
\times
\begin{pmatrix}
  x & x & x & x & x & x & x & x & x & x & x & x & x & x & \ldots \\
  x & x & x & x & x & x & x & x & x & x & x & x & x & x & \ldots \\
  x & x & x & x & x & x & x & x & x & x & x & x & x & x & \ldots
\end{pmatrix}
\]
Differentiation in 1d

1d differentiation matrix:

\[(\partial_x f)_i = \sum_{j=0}^{N-1} D_{ij} f_j\]  

(1)

sparse for finite differences

say 2 elements per row nonzero, \ldots 0 \ 0 \ -1 \ 0 \ 1 \ 0 \ 0 \ldots

full for pseudospectral, \approx N \ elements \ per \ row \ nonzero
Differentiation in 1d

1d differentiation matrix:

\[
(\partial_x f)_i = \sum_{j=0}^{N-1} D_{ij} f_j
\]  

(1)

sparse for finite differences  
say 2 elements per row nonzero, \ldots 0 0 -1 0 1 0 0 0 \ldots

full for pseudospectral, \approx N elements per row nonzero

Chebyshev in \( r \):

\[
CD_{ij} = \frac{c_i}{c_j} \frac{(-1)^{i+j}}{x_i - x_j} \quad \text{for } i \neq j, \quad CD_{ii} = -\sum_{j=0, j \neq i}^{N-1} CD_{ij}
\]

(2)

Fourier in \( \theta \) (doubled) and \( \phi \):

\[
FD_{ij} = \frac{(-1)^{i+j}}{2 \tan\left(\frac{x_i - x_j}{2}\right)} \quad \text{for } i \neq j, \quad FD_{ii} = 0
\]

(3)
Differentiation in 3d

\[ N = n_1 n_2 n_3 \] field values, matrices \( N \times N = n_1 n_2 n_3 \times n_1 n_2 n_3 \)

1d diff. matrices \( D_1 = C D_{n_1 \times n_1}, D_2 = F D_{n_2 \times n_2}, D_3 = F D_{n_3 \times n_3} \)

3d spectral diff. matrices are (semi-)sparse,
   e.g. \( n_1 \times n_1 n_2 n_3 \) nonzero, i.e. \( n^4 \) of \( n^6 \) nonzero
Differentiation in 3d

\( N = n_1 n_2 n_3 \) field values, matrices \( N \times N = n_1 n_2 n_3 \times n_1 n_2 n_3 \)

1d diff. matrices \( D_1 = CD_{n_1 \times n_1}, \quad D_2 = FD_{n_2 \times n_2}, \quad D_3 = FD_{n_3 \times n_3} \)

3d spectral diff. matrices are (semi-)sparse,

\[ \text{e.g. } n_1 \times n_1 n_2 n_3 \text{ nonzero, i.e. } n^4 \text{ of } n^6 \text{ nonzero} \]

implement as sparse matrices: elegant, say Trefethen/MATLAB (2000)

\[ D_{3d}^1 = D_1 \otimes I_2 \otimes I_3, \quad D_{3d}^2 = I_1 \otimes D_2 \otimes I_3, \quad D_{3d}^3 = I_1 \otimes I_2 \otimes D_3 \quad (4) \]
Differentiation in 3d

implement as sparse matrices: elegant, say Trefethen/MATLAB (2000)

\[
D_{3d}^3 = D_1 \otimes I_2 \otimes I_3, \quad D_{3d}^2 = I_1 \otimes D_2 \otimes I_3, \quad D_{3d}^1 = I_1 \otimes I_2 \otimes D_3
\]  

(5)

but: large overhead in MATLAB, Intel MKL; new/not tested: CUSPARSE

\[
\frac{\partial}{\partial n_1} u = (D_1)_{n_1 \times n_1} (u)_{n_1 \times n_2 n_3} \quad n_v = (n_1 n_2 n_3)
\]

\[
\frac{\partial}{\partial n_2} v = (D_2)_{n_2 \times n_2} (v)_{n_2 \times n_3 n_v} \quad n_1 = (n_1 n_2 n_3)
\]

\[
\frac{\partial}{\partial n_3} u = (D_3)_{n_3 \times n_3} (u)_{n_1 n_2 n_3} \quad n_v = (n_1 n_2 n_3)
\]

\[
\frac{\partial^2}{\partial n_1^2} v = ((\frac{\partial}{\partial n_2} v)_{n_2 \times n_3 n_v} \times n_1)_{n_1 \times n_1}
\]

\[
\frac{\partial^2}{\partial n_1 \partial n_2} u = ((\frac{\partial}{\partial n_2} v)_{n_2 \times n_3 n_v} \times n_1)_{n_1 \times n_2 n_3}
\]

faster than sparse-MM by more than factor 2 on CPU
Differentiation in 3d

implement as sparse matrices: elegant, say Trefethen/MATLAB (2000)

\[ D_{3d}^1 = D_1 \otimes I_2 \otimes I_3, \quad D_{3d}^2 = I_1 \otimes D_2 \otimes I_3, \quad D_{3d}^3 = I_1 \otimes I_2 \otimes D_3 \]  \hspace{1cm} (5)

but: large overhead in MATLAB, Intel MKL; new/not tested: CUSPARSE

want: matrix-matrix multiplication with strides
missing in BLAS and CUBLAS, to do: implement our own MM

simple alternative: 1d MM with transposes, which are low cost
Differentiation in 3d

implement as sparse matrices: elegant, say Trefethen/MATLAB (2000)

\[ D^{3d}_1 = D_1 \otimes I_2 \otimes I_3, \quad D^{3d}_2 = I_1 \otimes D_2 \otimes I_3, \quad D^{3d}_3 = I_1 \otimes I_2 \otimes D_3 \]  

(5)

but: large overhead in MATLAB, Intel MKL; new/not tested: CUSPARSE

want: matrix-matrix multiplication with strides

missing in BLAS and CUBLAS, to do: implement our own MM

simple alternative: 1d MM with transposes, which are low cost

1d MM from CUBLAS, transpose from CUDA SDK

direction 1:

\[ (\partial_1 u)_{n_1 \times n_2 n_3 n_v} = (D_1)_{n_1 \times n_1}(u)_{n_1 \times n_2 n_3 n_v}, \]  

(6)

direction 2:

\[ (v)_{n_2 n_3 n_v \times n_1} = (u_{n_1 \times n_2 n_3 n_v})^T, \]  

(7)

\[ (\partial_2 v)_{n_2 \times n_3 n_v n_1} = (D_2)_{n_2 \times n_2}(v)_{n_2 \times n_3 n_v n_1}, \]  

(8)

\[ (\partial_2 u)_{n_1 \times n_2 n_3 n_v} = ((\partial_2 v)_{n_2 n_3 n_v \times n_1})^T, \]  

(9)

faster than sparse-MM by more than factor 2 on CPU
GPU Computing

• numerical relativity often requires terabytes and teraflops
• get teraflops by GPU computing
• reduce flops and avoid terabytes by spectral method (!)
• spectral methods are dominated by derivatives and filters especially on GPU: MM for simplicity, efficiency, availability
GPU Computing

- numerical relativity often requires terabytes and teraflops
- get teraflops by GPU computing
- reduce flops and avoid terabytes by spectral method (!)
- spectral methods are dominated by derivatives and filters especially on GPU: MM for simplicity, efficiency, availability

- here: NVIDIA’s CUDA, CUBLAS
- here: single CPU core, single GPU with many cores
- independently of GPU:
  goal is maximally simple, cleanly defined computational kernel
NVIDIA hardware available in this project

<table>
<thead>
<tr>
<th>GPU</th>
<th>Available</th>
<th>Memory</th>
<th>Cost</th>
<th>Cores</th>
<th>GHz</th>
<th>FPUs</th>
<th>GB/s</th>
<th>GF/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX 285</td>
<td>1/2009</td>
<td>2.0 GB</td>
<td>(400)</td>
<td>240</td>
<td>1.5</td>
<td>1/8, 30</td>
<td>159</td>
<td>90</td>
</tr>
<tr>
<td>GTX 480</td>
<td>3/2010</td>
<td>1.5 GB</td>
<td>400</td>
<td>480</td>
<td>1.4</td>
<td>1/8, 60</td>
<td>177</td>
<td>170</td>
</tr>
<tr>
<td>GTX 580</td>
<td>11/2010</td>
<td>1.5 GB</td>
<td>400</td>
<td>512</td>
<td>1.54</td>
<td>1/8, 64</td>
<td>192</td>
<td>200</td>
</tr>
<tr>
<td>C2050</td>
<td>8/2010</td>
<td>2.7 ECC</td>
<td>1800</td>
<td>448</td>
<td>1.15</td>
<td>1/2, 224</td>
<td>144</td>
<td>515</td>
</tr>
<tr>
<td>M2070</td>
<td>11/2010</td>
<td>5.5 ECC</td>
<td>2700</td>
<td>448</td>
<td>1.15</td>
<td>1/2, 224</td>
<td>150</td>
<td>515</td>
</tr>
<tr>
<td>Q2000M</td>
<td>3/2011</td>
<td>2.0 GB</td>
<td>300</td>
<td>192</td>
<td>1.1</td>
<td>1/8, 24</td>
<td>29</td>
<td>52</td>
</tr>
<tr>
<td>Intel i7</td>
<td>3/2010</td>
<td>24 GB</td>
<td>2000</td>
<td>6</td>
<td>2.67</td>
<td>2/1, 12</td>
<td>13</td>
<td>64</td>
</tr>
</tbody>
</table>

- double precision since 7/2008
- scientific computing with cheap consumer cards? (no)
- GB/s often primary advantage over CPU for 3d simulations
- achievable GF/s vary wildly, 5% to 60%
- bottleneck CPU ↔ GPU at 5 GB/s ⇒ port everything to GPU
Example for MM versus 2xFFT

- $n_1 \times n_1$ times $n_1 \times (20 \times 20 \times 54)$
- FFT strongly size dependent, several $O(N \log N)$
- break-even for some “small” $n_1$, e.g. Fornberg 1998, Boyd 2001, …

CPU: $n_1 \lesssim 60$
Example for MM versus 2xFFT

- \( n_1 \times n_1 \) times \( n_1 \times (20 \times 20 \times 54) \)
- FFT strongly size dependent, several \( O(N \log N) \)
- break-even for some “small” \( n_1 \), e.g. Fornberg 1998, Boyd 2001, ...

- CPU: \( n_1 \lesssim 60 \),
- GPU: \( n_1 \lesssim 100 \)

B. Brügmann, University of Jena, SFB/Transregio 7

http://www.tpi.uni-jena.de
Example for MM versus 2xFFT

• $n_1 \times n_1 \times (20 \times 20 \times 54)$

• FFT strongly size dependent, several $O(N \log N)$

• break-even for some "small" $n_1$, e.g. Fornberg 1998, Boyd 2001, ...

• GPU: $n_1 \lesssim 100$

• GTX betters Tesla except at 64, 128
### GPU Performance: matrix multiplication

<table>
<thead>
<tr>
<th>MatMul</th>
<th>Gflop/s</th>
<th>40×20×20</th>
<th>40×40×40</th>
<th>60×60×54</th>
<th>64×64×54</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX285</td>
<td>2.3</td>
<td>33</td>
<td>43</td>
<td>52</td>
<td>70</td>
</tr>
<tr>
<td>GTX480</td>
<td>3.2rc1</td>
<td>–</td>
<td>101</td>
<td>88</td>
<td>163</td>
</tr>
<tr>
<td>GTX580</td>
<td>3.2</td>
<td>–</td>
<td>117</td>
<td>104</td>
<td>192</td>
</tr>
<tr>
<td>GTX580</td>
<td>4.0rc</td>
<td>77</td>
<td>99</td>
<td>94</td>
<td>177</td>
</tr>
<tr>
<td>C2050</td>
<td>3.1</td>
<td>48</td>
<td>64</td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>C2050</td>
<td>3.2rc1</td>
<td>–</td>
<td>108</td>
<td>103</td>
<td>284</td>
</tr>
<tr>
<td>M2070</td>
<td>3.2</td>
<td>–</td>
<td>109</td>
<td>104</td>
<td>294</td>
</tr>
</tbody>
</table>

- rather respectable w.r.t. theoretical peak
- consumer card GTX580 does very well, but low memory, no ECC
- high-end card M2070 needs more optimization
- small matrix bug on Fermi for newly optimized CUDA 3.2
  GTX stability? card reboot CUDA 4.1
### GPU Performance: spectral evolution of black hole

<table>
<thead>
<tr>
<th>Grid</th>
<th>GPU Algebra</th>
<th>GPU MatMul</th>
<th>GPU</th>
<th>CPU</th>
<th>CPU/GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>25 × 09 × 18</td>
<td>18%</td>
<td>80%</td>
<td>16s</td>
<td>165s</td>
<td>10.0</td>
</tr>
<tr>
<td>37 × 15 × 30</td>
<td>25%</td>
<td>74%</td>
<td>49s</td>
<td>823s</td>
<td>16.7</td>
</tr>
<tr>
<td>49 × 21 × 42</td>
<td>23%</td>
<td>77%</td>
<td>133s</td>
<td>2903s</td>
<td>21.8</td>
</tr>
</tbody>
</table>

- GTX580 CUDA 4.0rc compared to one core of a i7-870
- 1000 RK4 evolution steps, no startup time
- matrix multiplications of derivatives and filter at 75% (well optimized)
- algebra in RHS of Einstein equations: 10,000 flop per time step
GPU Performance: spectral evolution of black hole

<table>
<thead>
<tr>
<th>Grid</th>
<th>GPU Algebra</th>
<th>GPU MatMul</th>
<th>GPU</th>
<th>CPU</th>
<th>CPU/GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>$25 \times 09 \times 18$</td>
<td>18%</td>
<td>80%</td>
<td>16s</td>
<td>165s</td>
<td>10.0</td>
</tr>
<tr>
<td>$37 \times 15 \times 30$</td>
<td>25%</td>
<td>74%</td>
<td>49s</td>
<td>823s</td>
<td>16.7</td>
</tr>
<tr>
<td>$49 \times 21 \times 42$</td>
<td>23%</td>
<td>77%</td>
<td>133s</td>
<td>2903s</td>
<td>21.8</td>
</tr>
</tbody>
</table>

GPU implementation is about 20x faster than single core CPU

MM on four CPU cores scaled only at 50% - 65% in Intel MKL

$\rightarrow$ GPU estimated 7x to 11x faster than quad core CPU
Summary

Results

• new algorithm to reduce “everything” to matrix multiplication
• small-matrix multiplications (or small FFTs) need development
• 20 times faster on GPU than on one CPU core
Summary

Results

• new algorithm to reduce “everything” to matrix multiplication
• small-matrix multiplications (or small FFTs) need development
• 20 times faster on GPU than on one CPU core

Goals

• multiple grids: MPI/GPU hybrid
• longevity of GPU computing: OpenCl? . . . ?
• independently of GPU: want maximally simple, cleanly defined computational kernels