Master Thesis Presentation

Simulating the gravitational field of a non-rotating neutron star on GPUs

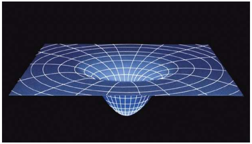
Tantilian Gkratsia

Aristotle University of Thessaloniki
&
Eberhard Karls Univeristat Tübingen

Supervisors: K. Kokkotas, N. Stergioulas, B. Zink

Contents

- Theoretical Introduction
  - The physical problem
  - CFC - approximation
- Computational Physics
  - Elliptic problems and multigrid
  - GPU programming
  - Optimization
- Results



The physical problem

- Simulating the dynamics of a relativistic rotating neutron star (on GPUs)
- ADM 3+1 formalism, Cartesian coordinates: $ds^2 = -(N^2 - \beta_i\beta^i)dt^2 + 2\beta_i dx^i dt + \gamma_{ij}dx^i dx^j$
- N: lapse function, β: spacelike shift three-vector, γ: three-metric
- In our gauge choice: $\beta_i = 0$
- $ds^2 = -N^2 dt^2 + \gamma_{ij}dx^i dx^j$

$$g_{\mu\nu} = \begin{pmatrix} -N^2 & 0 & 0 & 0 \\ 0 & \gamma_{11} & \gamma_{12} & \gamma_{13} \\ 0 & \gamma_{21} & \gamma_{22} & \gamma_{33} \\ 0 & \gamma_{31} & \gamma_{22} & \gamma_{33} \end{pmatrix}$$

The physical problem

- Simulating the dynamics of a relativistic rotating neutron star (on GPUs)
- ADM 3+1 formalism, Cartesian coordinates:
  $ds^2 = -(N^2 - \beta_i \beta^i)dt^2 + 2\beta_i dx^i dt + \gamma_{ij} dx^i dx^j$
- N: lapse function, β: spacelike shift three-vector, γ: three-metric
- In our gauge choice: $\beta_i = 0$
- $ds^2 = -N^2 dt^2 + \gamma_{ij} dx^i dx^j$

$$g_{\mu\nu} = \begin{pmatrix} -N^2 & 0 & 0 & 0 \\ 0 & \gamma_{11} & \gamma_{12} & \gamma_{13} \\ 0 & \gamma_{21} & \gamma_{22} & \gamma_{33} \\ 0 & \gamma_{31} & \gamma_{22} & \gamma_{33} \end{pmatrix}$$

The physical problem

- Simulating the dynamics of a relativistic rotating neutron star (on GPUs)
- ADM 3+1 formalism, Cartesian coordinates:
  $ds^2 = -(N^2 - \beta_i \beta^i) dt^2 + 2\beta_i dx^i dt + \gamma_{ij} dx^i dx^j$
- N: lapse function, β: spacelike shift three-vector, γ: three-metric
- In our gauge choice: $\beta_i = 0$
- $ds^2 = -N^2 dt^2 + \gamma_{ij} dx^i dx^j$

$$g_{\mu\nu} = \begin{pmatrix} -N^2 & 0 & 0 & 0 \\ 0 & \gamma_{11} & \gamma_{12} & \gamma_{13} \\ 0 & \gamma_{21} & \gamma_{22} & \gamma_{33} \\ 0 & \gamma_{31} & \gamma_{22} & \gamma_{33} \end{pmatrix}$$

The physical problem

- Simulating the dynamics of a relativistic rotating neutron star (on GPUs)
- ADM 3+1 formalism, Cartesian coordinates:
  $ds^2 = -(N^2 - \beta_i \beta^i)dt^2 + 2\beta_i dx^i dt + \gamma_{ij} dx^i dx^j$
- N: lapse function, β: spacelike shift three-vector, γ: three-metric
- In our gauge choice: $\beta_i = 0$
- $ds^2 = -N^2 dt^2 + \gamma_{ij} dx^i dx^j$

$$g_{\mu\nu} = \begin{pmatrix} -N^2 & 0 & 0 & 0 \\ 0 & \gamma_{11} & \gamma_{12} & \gamma_{13} \\ 0 & \gamma_{21} & \gamma_{22} & \gamma_{33} \\ 0 & \gamma_{31} & \gamma_{22} & \gamma_{33} \end{pmatrix}$$

The physical problem

- Simulating the dynamics of a relativistic rotating neutron star (on GPUs)
- ADM 3+1 formalism, Cartesian coordinates: $ds^2 = -(N^2 - \beta_i \beta^i) dt^2 + 2\beta_i dx^i dt + \gamma_{ij} dx^i dx^j$
- N: lapse function, β: spacelike shift three-vector, γ: three-metric
- In our gauge choice: $\beta_i = 0$
- $ds^2 = -N^2 dt^2 + \gamma_{ij} dx^i dx^j$

$$g_{\mu\nu} = \begin{pmatrix} -N^2 & 0 & 0 & 0 \\ 0 & \gamma_{11} & \gamma_{12} & \gamma_{13} \\ 0 & \gamma_{21} & \gamma_{22} & \gamma_{33} \\ 0 & \gamma_{31} & \gamma_{22} & \gamma_{33} \end{pmatrix}$$

CFC - approximation

- Conformal Flatness Condition: $\gamma_{ij} = \phi^4 \eta_{ij}$, ϕ: conformal factor, η_{ij}: flat space metric.
- With ADM 3+1 formalism and CFC, we get the Einstein equation in the form[1]:
  - $\nabla^2 \phi = -2\pi\phi^5 \left(\rho h W^2 - P + \frac{K_{ij}K^{ij}}{16\pi} \right)$
  - $\nabla^2 (N\phi) = 2\pi N\phi^5 \left(\rho h (3W^2 - 2) + 5P + \frac{7K_{ij}K^{ij}}{16\pi} \right)$
  - $\nabla^2 \beta^i = 16\pi N \phi^4 S^i + 2\hat{K}^{ij}\hat{\nabla}_j \left(\frac{N}{\phi^6} \right) - \frac{1}{3}\hat{\nabla}^i \hat{\nabla}_k \beta^k$

[1] Relativistic simulations of rotational core collapse I. Methods, initial models, and code tests H. Dimmelmeier, J.A. Font, and E. Müller, A&A 388, 917 - 935 (2002)

CFC - approximation

- Conformal Flatness Condition: $\gamma_{ij} = \phi^4 \eta_{ij}$, ϕ: conformal factor, η_{ij}: flat space metric.
- With ADM 3+1 formalism and CFC, we get the Einstein equation in the form[1]:
  - $\nabla^2 \phi = -2\pi\phi^5 \left(\rho h W^2 - P + \frac{K_{ij}K^{ij}}{16\pi} \right)$
  - $\nabla^2 (N\phi) = 2\pi N \phi^5 \left(\rho h (3W^2 - 2) + 5P + \frac{7K_{ij}K^{ij}}{16\pi} \right)$
  - $\nabla^2 \beta^i = 16\pi N \phi^4 S^i + 2\hat{K}^{ij} \hat{\nabla}_j \left(\frac{N}{\phi^6} \right) - \frac{1}{3} \hat{\nabla}^i \hat{\nabla}_k \beta^k$

[1] Relativistic simulations of rotational core collapse I. Methods, initial models, and code tests H. Dimmelmeier, J.A. Font, and E. Müller, A&A 388, 917 - 935 (2002)

CFC $\Rightarrow$ Elliptic equations

Where:

ϕ: conformal factor, ρ : rest mass density, $h = 1 + \varepsilon P/\rho$: specific relativistic enthalpy, P: pressure, $W = Nu^t$ (Lorentz factor)

For a non-rotating star: $K_{ij} = \mathcal{L}_n \gamma_{ij} = 0$, $\beta_i = 0$, $w = 1$

So our problem is to solve the non-linear elliptic (Poisson-like) equations:

$$\nabla^2 \phi = -2\pi\phi^5 \left(\rho h W^2 - P\right)$$

$$\nabla^2 (N\phi) = 2\pi N\phi^5 \left(\rho h (3W^2 - 2) + 5P\right)$$

CFC $\Rightarrow$ Elliptic equations

Where:

ϕ: conformal factor, ρ : rest mass density, $h = 1 + \varepsilon P/\rho$: specific relativistic enthalpy, P: pressure, $W = Nu^t$ (Lorentz factor)

For a non-rotating star: $K_{ij} = \mathcal{L}_n \gamma_{ij} = 0$, $\beta_i = 0$, $w = 1$

So our problem is to solve the non-linear elliptic (Poisson-like) equations:

$$\nabla^2 \phi = -2\pi\phi^5 \left(\rho h W^2 - P\right)$$

$$\nabla^2 (N\phi) = 2\pi N \phi^5 \left(\rho h (3W^2 - 2) + 5P\right)$$

CFC $\Rightarrow$ Elliptic equations

Where:
ϕ: conformal factor, ρ : rest mass density, $h = 1 + \varepsilon P/\rho$: specific relativistic enthalpy, P: pressure, $W = Nu^t$ (Lorentz factor)
For a non-rotating star: $K_{ij} = \mathcal{L}_n \gamma_{ij} = 0$, $\beta_i = 0$, $w = 1$
So our problem is to solve the non-linear elliptic (Poisson-like) equations:

$$\nabla^2 \phi = -2\pi\phi^5 \left(\rho h W^2 - P\right)$$

$$\nabla^2 (N\phi) = 2\pi N\phi^5 \left(\rho h (3W^2 - 2) + 5P\right)$$

Elliptic Solvers

Methods of solving elliptic equations:

- Iterative (Gauss-Seidel,Jacobi):

$$\left.\begin{array}{l} \nabla^2 u = s \\ \nabla^2 u \approx \frac{u_{i+1}+u_{i-1}-2u_i}{\Delta x^2} \end{array}\right\} \Rightarrow \frac{u_{i+1}+u_{i-1}-2u_i}{\Delta x^2} = s$$

$$\Rightarrow u_i = \frac{1}{2}\left(u_{i-1} + u_{i+1} - s \cdot \Delta x^2\right)$$

- Conjugate Gradient
- Other methods (Spectral,...)

Elliptic Solvers

Methods of solving elliptic equations:

- Iterative (Gauss-Seidel,Jacobi):

$$\left.\begin{array}{l}\nabla^2 u = s \\ \nabla^2 u \approx \frac{u_{i+1}+u_{i-1}-2u_i}{\Delta x^2}\end{array}\right\} \Rightarrow \frac{u_{i+1}+u_{i-1}-2u_i}{\Delta x^2} = s$$

$$\Rightarrow u_i = \frac{1}{2}\left(u_{i-1} + u_{i+1} - s \cdot \Delta x^2\right)$$

- Conjugate Gradient
- Other methods (Spectral,...)

Elliptic Solvers

Methods of solving elliptic equations:

- Iterative (Gauss-Seidel,Jacobi):

$$\left.\begin{array}{l} \nabla^2 u = s \\ \nabla^2 u \approx \frac{u_{i+1}+u_{i-1}-2u_i}{\Delta x^2} \end{array}\right\} \Rightarrow \frac{u_{i+1}+u_{i-1}-2u_i}{\Delta x^2} = s$$

$$\Rightarrow u_i = \frac{1}{2}\left(u_{i-1} + u_{i+1} - s \cdot \Delta x^2\right)$$

- Conjugate Gradient
- Other methods (Spectral,...)

Elliptic Solvers

Methods of solving elliptic equations:

- Iterative (Gauss-Seidel,Jacobi):

$$\left.\begin{array}{l} \nabla^2 u = s \\ \nabla^2 u \approx \frac{u_{i+1}+u_{i-1}-2u_i}{\Delta x^2} \end{array}\right\} \Rightarrow \frac{u_{i+1}+u_{i-1}-2u_i}{\Delta x^2} = s$$

$$\Rightarrow u_i = \frac{1}{2}\left(u_{i-1} + u_{i+1} - s \cdot \Delta x^2\right)$$

- Conjugate Gradient
- Other methods (Spectral,...)

Multigrid

A technique to speed up an iterative solver

- The equation: $Au = f$
- The residual: $r = f - Au$
- The exact solution $v = u + e$, e: the error
- Using Gauss-Seidel method, high frequency errors are eliminated faster than low frequency errors
- If the error is distributed in a low frequency mode, the convergence rate is slow

More about Multigrid: A Multigrid Tutorial, Second Edition
by William L. Briggs, Van Emden Henson, Steve F. McCormic

Multigrid

A technique to speed up an iterative solver

- The equation: $Au = f$
- The residual: $r = f - Au$
- The exact solution $v = u + e$, e: the error
- Using Gauss-Seidel method, high frequency errors are eliminated faster than low frequency errors
- If the error is distributed in a low frequency mode, the convergence rate is slow

More about Multigrid: A Multigrid Tutorial, Second Edition
by William L. Briggs, Van Emden Henson, Steve F. McCormic

Multigrid

A technique to speed up an iterative solver

- The equation: $Au = f$
- The residual: $r = f - Au$
- The exact solution $v = u + e$, e: the error
- Using Gauss-Seidel method, high frequency errors are eliminated faster than low frequency errors
- If the error is distributed in a low frequency mode, the convergence rate is slow

More about Multigrid: A Multigrid Tutorial, Second Edition
by William L. Briggs, Van Emden Henson, Steve F. McCormic

Multigrid

A technique to speed up an iterative solver

- The equation: $Au = f$
- The residual: $r = f - Au$
- The exact solution $v = u + e$, e: the error
- Using Gauss-Seidel method, high frequency errors are eliminated faster than low frequency errors
- If the error is distributed in a low frequency mode, the convergence rate is slow

More about Multigrid: A Multigrid Tutorial, Second Edition
by William L. Briggs, Van Emden Henson, Steve F. McCormic

Multigrid

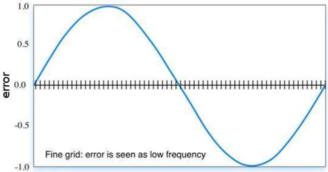
A technique to speed up an iterative solver

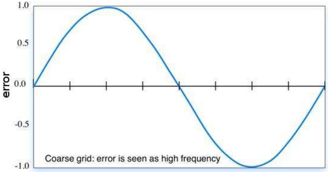
- The equation: $Au = f$
- The residual: $r = f - Au$
- The exact solution $v = u + e$, e: the error
- Using Gauss-Seidel method, high frequency errors are eliminated faster than low frequency errors
- If the error is distributed in a low frequency mode, the convergence rate is slow

More about Multigrid: A Multigrid Tutorial, Second Edition
by William L. Briggs, Van Emden Henson, Steve F. McCormic

Multigrid - grid resolution

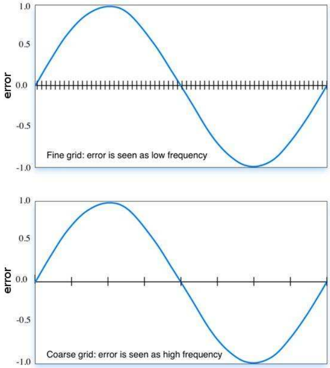
- By reducing the grid resolution, low frequency errors appear as high frequency errors
- On the coarser level we solve for the error, using Gauss-Seidel method
- After finding the error, we go to the finer level and correct the solution





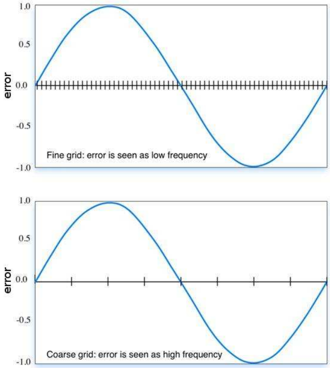
Multigrid - grid resolution

- By reducing the grid resolution, low frequency errors appear as high frequency errors
- On the coarser level we solve for the error, using Gauss-Seidel method
- After finding the error, we go to the finer level and correct the solution



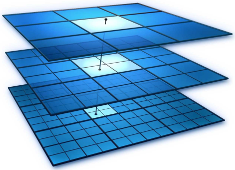
Multigrid - grid resolution

- By reducing the grid resolution, low frequency errors appear as high frequency errors
- On the coarser level we solve for the error, using Gauss-Seidel method
- After finding the error, we go to the finer level and correct the solution



Multigrid - the idea

- If $v = u + e$ is the exact solution, then:



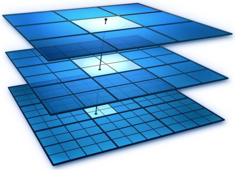
$$Av = f \Rightarrow A(u + e) = f \Rightarrow$$
$$Au + Ae = f \Rightarrow Ae = f - Au \Rightarrow$$
$$\boxed{Ae = r}$$

- We solve for the error on the coarser level using the residual as the source function
- Restriction operation: interpolation method used to inject the residual from a fine grid to the source of the coarser grid
- Correction operation: interpolation to the finer level and correction of the solution

Multigrid - the idea

- If $v = u + e$ is the exact solution, then:

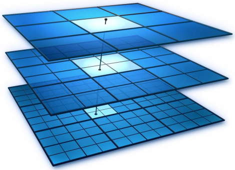
$$Av = f \Rightarrow A(u+e) = f \Rightarrow$$
$$Au + Ae = f \Rightarrow Ae = f - Au \Rightarrow$$
$$\boxed{Ae = r}$$



- We solve for the error on the coarser level using the residual as the source function
- Restriction operation: interpolation method used to inject the residual from a fine grid to the source of the coarser grid
- Correction operation: interpolation to the finer level and correction of the solution

Multigrid - the idea

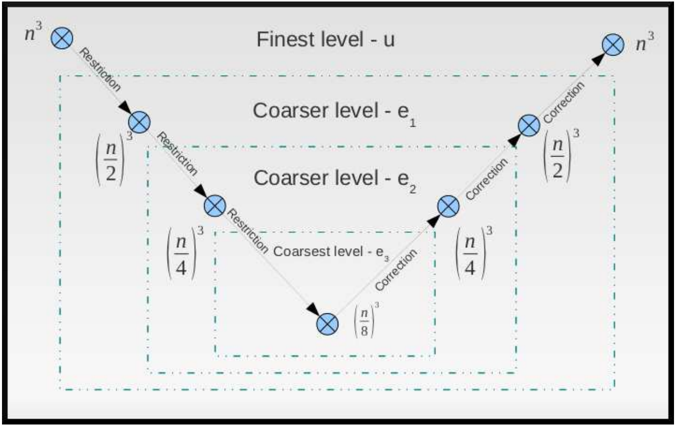
- If $v = u + e$ is the exact solution, then:



$$Av = f \Rightarrow A(u+e) = f \Rightarrow$$
$$Au + Ae = f \Rightarrow Ae = f - Au \Rightarrow$$
$$\boxed{Ae = r}$$

  - We solve for the error on the coarser level using the residual as the source function
  - Restriction operation: interpolation method used to inject the residual from a fine grid to the source of the coarser grid
  - Correction operation: interpolation to the finer level and correction of the solution

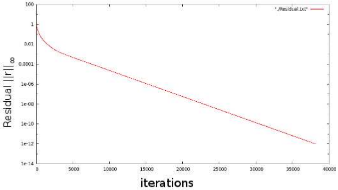
Multigrid - the V-cycle



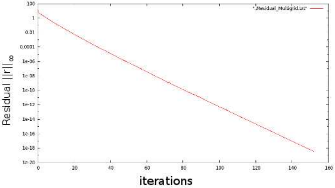
Multigrid - speed up

Convergence rate of Gauss - Seidel with and without *Multigrid*

$$\nabla^2\Phi = 4\pi\rho$$



Convergence without *Multigrid*



Convergence with *Multigrid*

GPU - advantages

- We need faster computer systems
- CPUs are close to the limit (overheating, quantum effects,...)
- Solution: Parallel computing
- GPU: Graphics Processor Unit
  - Designed for parallel processing 3D graphics
  - Many processing cores on a device (e.g. 480)
  - More transistors are devoted to computation than for control logic and caches

GPU - advantages

- We need faster computer systems
- CPUs are close to the limit (overheating, quantum effects,...)
- Solution: Parallel computing
- GPU: Graphics Processor Unit
  - Designed for parallel processing 3D graphics
  - Many processing cores on a device (e.g. 480)
  - More transistors are devoted to computation than for control logic and caches

GPU - advantages

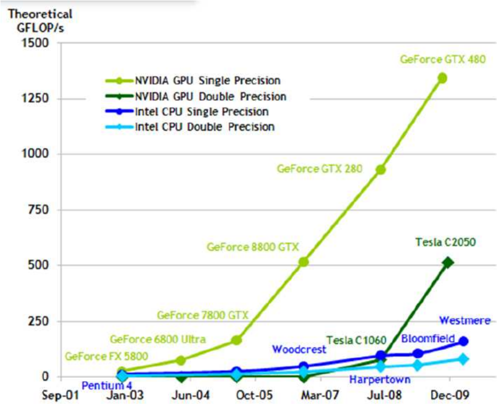
- We need faster computer systems
- CPUs are close to the limit (overheating, quantum effects,...)
- Solution: Parallel computing
- GPU: Graphics Processor Unit
  - Designed for parallel processing 3D graphics
  - Many processing cores on a device (e.g. 480)
  - More transistors are devoted to computation than for control logic and caches

GPU - advantages

- We need faster computer systems
- CPUs are close to the limit (overheating, quantum effects,...)
- Solution: Parallel computing
- GPU: Graphics Processor Unit
  - Designed for parallel processing 3D graphics
  - Many processing cores on a device (e.g. 480)
  - More transistors are devoted to computation than for control logic and caches

GPU - peak performance

Peak performance of GPUs



GPU - CUDA

- CUDA (Compute Unified Device Architecture) is a parallel computing language developed by NVIDIA
- Programming in C/C++ environment with additional keycodes

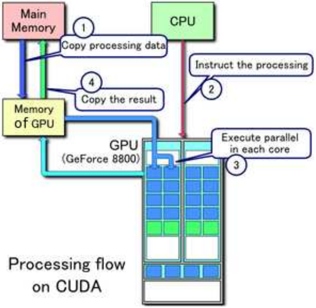
```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
int i = threadIdx.x;
C[i] = A[i] + B[i];
}

// Main function
int main()
{
...
// Kernel invocation with N threads
VecAdd<<<1, N>>>(A, B, C);
}
```

GPU - CUDA

Processing flow from computer to graphic card, in CUDA

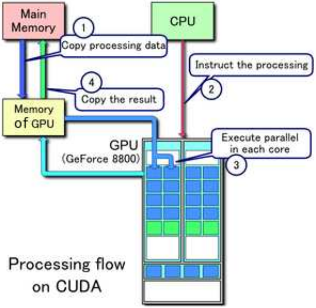
- Copy Processing data from main memory to GPU memory
- Execute parallel in each core
- Copy data back to the main memory (for the outputs)



GPU - CUDA

Processing flow from computer to graphic card, in CUDA

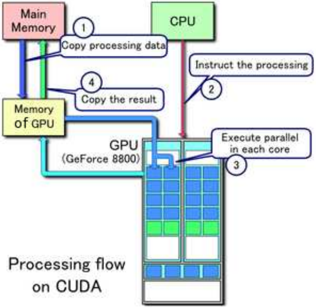
- Copy Processing data from main memory to GPU memory
- Execute parallel in each core
- Copy data back to the main memory (for the outputs)



GPU - CUDA

Processing flow from computer to graphic card, in CUDA

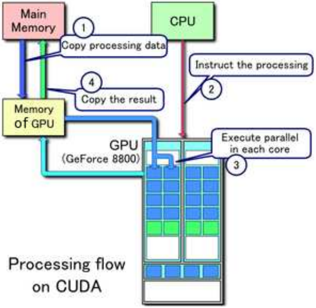
- Copy Processing data from main memory to GPU memory
- Execute parallel in each core
- Copy data back to the main memory (for the outputs)



GPU - CUDA

Processing flow from computer to graphic card, in CUDA

- Copy Processing data from main memory to GPU memory
- Execute parallel in each core
- Copy data back to the main memory (for the outputs)



GPU vs CPU

Solving 3D elliptic equation (Poisson) with Gauss - Seidel method

- Without multigrid (3000 iter. - 35000 needed):
  - Duration on CPU: 374.96 s
  - Duration on GPU: 57.48 s
- With multigrid (120 iter. - 90 needed):
  - Duration on CPU: 8.91 s
  - Duration on GPU: 1.18 s

Speed up: $\sim$ 7.55x
(Without optimization)

GPU vs CPU

Solving 3D elliptic equation (Poisson) with Gauss - Seidel method

- Without multigrid (3000 iter. - 35000 needed):
  - Duration on CPU: 374.96 s
  - Duration on GPU: 57.48 s
- With multigrid (120 iter. - 90 needed):
  - Duration on CPU: 8.91 s
  - Duration on GPU: 1.18 s

Speed up: $\sim$ 7.55x
(Without optimization)

GPU vs CPU

Solving 3D elliptic equation (Poisson) with Gauss - Seidel method

- Without multigrid (3000 iter. - 35000 needed):
  - Duration on CPU: 374.96 s
  - Duration on GPU: 57.48 s
- With multigrid (120 iter. - 90 needed):
  - Duration on CPU: 8.91 s
  - Duration on GPU: 1.18 s

Speed up: $\sim$ 7.55x
(Without optimization)

GPU vs CPU

Solving 3D elliptic equation (Poisson) with Gauss - Seidel method

- Without multigrid (3000 iter. - 35000 needed):
  - Duration on CPU: 374.96 s
  - Duration on GPU: 57.48 s
- With multigrid (120 iter. - 90 needed):
  - Duration on CPU: 8.91 s
  - Duration on GPU: 1.18 s

Speed up: $\sim$ 7.55x
(Without optimization)

CUDA - optimization

Basic optimizations:

- Optimized algorithms:
  - Maximizing independent parallelism
  - Sometimes it's better to recompute than to cache
  - More computations on GPU to avoid data transfers to the Host
- Memory optimization
  - Local and Shared memory
  - Using Shared memory
  - Bank conflicts
- Maximizing multiprocessor usage

CUDA - optimization

Basic optimizations:

- Optimized algorithms:
  - Maximizing independent parallelism
  - Sometimes it's better to recompute than to cache
  - More computations on GPU to avoid data transfers to the Host
- Memory optimization
  - Local and Shared memory
  - Using Shared memory
  - Bank conflicts
- Maximizing multiprocessor usage

CUDA - optimization

Basic optimizations:

- Optimized algorithms:
  - Maximizing independent parallelism
  - Sometimes it's better to recompute than to cache
  - More computations on GPU to avoid data transfers to the Host
- Memory optimization
  - Local and Shared memory
  - Using Shared memory
  - Bank conflicts
- Maximizing multiprocessor usage

CUDA - optimization

Basic optimizations:

- Optimized algorithms:
  - Maximizing independent parallelism
  - Sometimes it's better to recompute than to cache
  - More computations on GPU to avoid data transfers to the Host
- Memory optimization
  - Local and Shared memory
  - Using Shared memory
  - Bank conflicts
- Maximizing multiprocessor usage

Back to our simulation

- Solving the (non-linear) equations:

$$\nabla^2 \phi = -2\pi\phi^5 \left(\rho h W^2 - P\right)$$

$$\nabla^2 (N\phi) = 2\pi N\phi^5 \left(\rho h (3W^2 - 2) + 5P\right)$$

- Using Gauss - Seidel method
- With multigrid technique
- Implemented in CUDA

Back to our simulation

- Solving the (non-linear) equations:

$$\nabla^2\phi = -2\pi\phi^5\left(\rho h W^2 - P\right)$$

$$\nabla^2(N\phi) = 2\pi N\phi^5\left(\rho h(3W^2 - 2) + 5P\right)$$

- Using Gauss - Seidel method
- With multigrid technique
- Implemented in CUDA

Back to our simulation

- Solving the (non-linear) equations:

$$\nabla^2\phi = -2\pi\phi^5\left(\rho h W^2 - P\right)$$

$$\nabla^2(N\phi) = 2\pi N\phi^5\left(\rho h(3W^2 - 2) + 5P\right)$$

- Using Gauss - Seidel method
- With multigrid technique
- Implemented in CUDA

Back to our simulation

- Solving the (non-linear) equations:

$$\nabla^2\phi = -2\pi\phi^5\left(\rho h W^2 - P\right)$$

$$\nabla^2(N\phi) = 2\pi N\phi^5\left(\rho h(3W^2 - 2) + 5P\right)$$

- Using Gauss - Seidel method
- With multigrid technique
- Implemented in CUDA

Facing the non-linearity

$$\nabla^2\phi = -2\pi\phi^5\left(\rho h W^2 - P\right)$$

- Initial guess for ϕ in the right hand side (rhs) of the equation
- Use the solution to replace the ϕ in the rhs and then solve again
- Repeat until we reach the desired accuracy

```
// Initialization
Init(phi,....);
// Outer loop
for(int i...){
   // initialize source with new phi
   Init(phi,....);
   // full V-cycle
   for(int...){
      . . .
   }
}
```

Facing the non-linearity

$$\nabla^2\phi = -2\pi\phi^5\left(\rho h W^2 - P\right)$$

- Initial guess for ϕ in the right hand side (rhs) of the equation
- Use the solution to replace the ϕ in the rhs and then solve again
- Repeat until we reach the desired accuracy

```
// Initialization
Init(phi,....);
// Outer loop
for(int i...){
    // initialize source with new phi
    Init(phi,....);
    // full V-cycle
    for(int...){
        . . .
    }
}
```

Facing the non-linearity

$$\nabla^2\phi = -2\pi\phi^5\left(\rho h W^2 - P\right)$$

- Initial guess for ϕ in the right hand side (rhs) of the equation
- Use the solution to replace the ϕ in the rhs and then solve again
- Repeat until we reach the desired accuracy

```
// Initialization
Init(phi,....);
// Outer loop
for(int i...){
    // initialize source with new phi
    Init(phi,....);
    // full V-cycle
    for(int...){
        . . .
    }
}
```

Facing the non-linearity

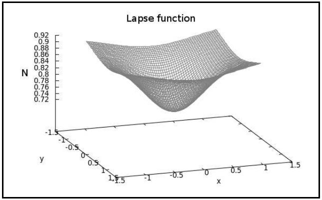
$$\nabla^2 \phi = -2\pi\phi^5 \left(\rho h W^2 - P\right)$$

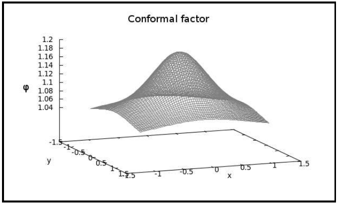
- Initial guess for ϕ in the right hand side (rhs) of the equation
- Use the solution to replace the ϕ in the rhs and then solve again
- Repeat until we reach the desired accuracy

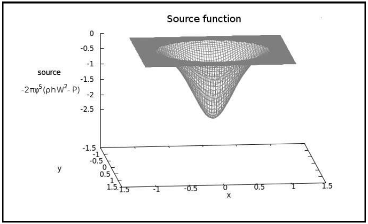
```
// Initialization
Init(phi,....);
// Outer loop
for(int i...){
    // initialize source with new phi
    Init(phi,....);
    // full V-cycle
    for(int...){
        . . .
    }
}
```

Solution

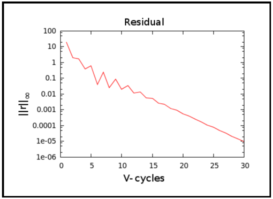
Equatorial plane of the solutions (z=0)







Residual ($\| r \|_\infty$)

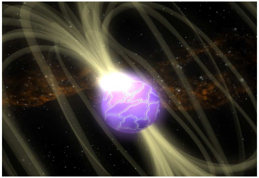


- Iterations: 180
- Duration (for for a grid size of 65^3 and on a double precision device with 480 CUDA-cores) : *8.96 s*
- Duration (for a grid size of 65^3 and on GTX 460 -single precision- with 336 CUDA-cores) :
  - Without optimization: *2.75 s*
  - With optimization: *0.47 s*

Thank you

Thank you for your time

Next step: rotating neutron star



References:

- NVIDIA (www.nvidia.com)
- CUDA BY EXAMPLE - J. Sanders and E. Kandrot
- Spacetime and Geometry - S.M. Carroll
- Relativistic simulations of rotational core collapse I. Methods, initial models, and code tests H. Dimmelmeier, J.A. Font, and E. Müller, A&A 388, 917 - 935 (2002)

- Special thanks to Dr. Burkhard Zink for his useful advices and for the inspiration.