SECRET SUPER COMPUTERS

Justin McKennon
Senior Electrical Engineer
NTS Lightning Technologies

HISTORY OF PROBLEM SOLVING

- When you think of all the great inventions, theorems, equations and ideas throughout history, what do they have in common?
- Each and every one of them was developed for the same purpose: There existed a problem that did not yet have a solution, or the "accepted" solution did not sit well with someone (Einstein)
- As time has gone on, the complexity of these unsolved problems has and will continue to increase
  - We've picked most of the low hanging fruit already!

HISTORY OF PROBLEM SOLVING

- As the complexity of the problems has increased, it became more and more difficult to develop solutions
- The invention of computers has drastically increased the ability of people to solve problems
  - Calculators, cell phones, video games
- Despite the increasing complexity of problems, computers have allowed mathematicians, scientists, engineers, businessmen etc… to make considerable strides and innovations in every aspect of their respective areas that would have required inordinate amounts of data processing

HISTORY OF PROBLEM SOLVING

HISTORY OF PROBLEM SOLVING

- Computers have actually accelerated the complexity of modern day problems and their solutions
  - More powerful hardware means bigger problems (variables) and more data
- But what happens when the complexity reaches levels that even CPUs cannot efficiently solve the problems?
- To answer this, I will need to introduce you to someone

COMPUTING PITFALLS

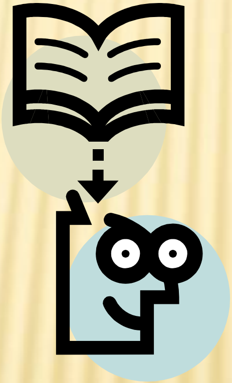
- This is Clip Art Jim. Throughout this talk he will continue to come up to help me to explain some of the ideas here
- Jim is analogous to a single core computer
- Jim has no trouble solving standard problems

COMPUTING PITFALLS

- Now, as the difficulty and complexity of the problems increases, the time it takes for Jim to solve problems significantly increases
- Jim is not as efficient of a problem solver when it comes to tough problems

COMPUTING PITFALLS

- So what does Jim do to solve these more difficult problems?
- Easy! Get smarter!



COMPUTING PITFALLS

- In the world of computers, the efficiency of a single computer (CPU) with respect to calculations and problem solving can be improved by increasing the speed of its processor (clock speed)
- As the difficulty of the problems increases, CPU manufacturers will have to crank of the processor speeds to maintain problem solving and task efficiency

COMPUTING PITFALLS

- Now, Jim is smarter. Whenever the progression of problem complexity makes him an inefficient problem solver, Jim simply makes himself smarter
- This cycle of Jim increasing his intelligence when he becomes inefficient continues over and over again… that is, until a critical point is reached
- What happens when Jim just can't get any smarter (think of his brain simply being full)?

COMPUTING PITFALLS

- In the computing industry, this is known as hitting a frequency wall
- When this wall is reached, CPU clock speeds begin to stall out
- This typically occurs around 3.5-4 GHz

COMPUTING PITFALLS

- Computer manufacturers typically add more transistors to a CPU to increase its' speed and performance
- As more and more transistors are added the law of diminishing returns begins to take hold
- More transistors yield less and less of a performance gain
- Increasing the number of transistors also increases the power consumption and heat dissipation of the CPU
- Since CPU sizes have largely remained the same in recent years, the transistors inside of them have had to become much smaller

COMPUTING PITFALLS

- Near the frequency wall, adding more transistors doesn’t yield any worthwhile performance gain
- Additionally, increasing the number of transistors increases power consumption and heat dissipation as well
- Too many transistors will make the CPU too hot and consume to much power to be worth using
  - You’ll need crazy expensive nitrogen cooling systems, or something similar
- However, this performance bottleneck does not stop the ever increasing difficulty/complexity of problems
- What to do?

COMPUTING PITFALLS

- Computer industry has solved this performance bottleneck by moving to the modern, many-core architecture
- By having more than one processor (core) in a computer, the workload can be divided and the performance will increase!
- Most of the computers in use today utilize this style

COMPUTING PITFALLS

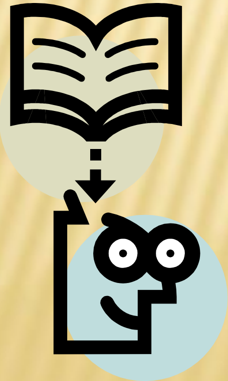
- Jim is now part of a team
- His team is able to solve the problems that handcuffed Jim by himself with relative ease!

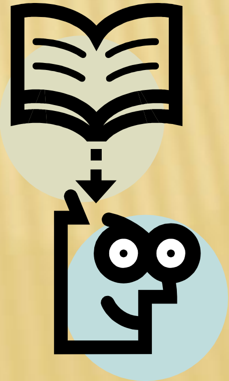
COMPUTING PITFALLS

So that’s it, right?

COMPUTING PITFALLS

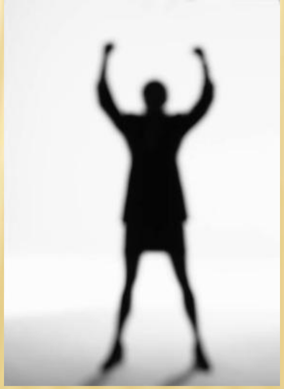
- Same cycle as before continues on
- Problems get tougher -> team gets smarter (speeds increase) -> problems get tougher -> team cannot get any smarter, so more team members are added to restore efficiency
- There is a very finite limit to the amount of team members that Jim can work with





COMPUTING PITFALLS

- Physical limitations now prevent the addition of more cores
- Space – how BIG of a computer do you really want!?
- Power consumption and heat dissipation become issues as well
- The computing industry is rapidly approaching this



COMPUTER PITFALLS

- Jim and his team are already as big and as smart as they can get
- It takes hours… even days to get meaningful results
- Corners are forced to be cut due to performance constraints and the accuracy of results plummets with complex problems
- What now?
- This is the question currently staring at the computing industry
- Can nothing be done?

THE SECRET SUPERCOMPUTER

- In the early 2000s, NVIDIA had one of the most significant scientific breakthroughs in the past decade
- Enter the realm of the GPU!

GPUS

- Pause on our technological timeline for a moment
- If you've ever played any sort of graphics based computer game: World of Warcraft, The Sims, sports games etc… you're already subconsciously aware of the power of GPUs
- In-game physics, explosions, landscapes, characters… all of these are rendered by the GPU via hundreds and hundreds of thousands of calculations every single second
- GPUs contain hundreds (sometimes thousands) of cores, making them ideal for tasks such as these!

GPUS

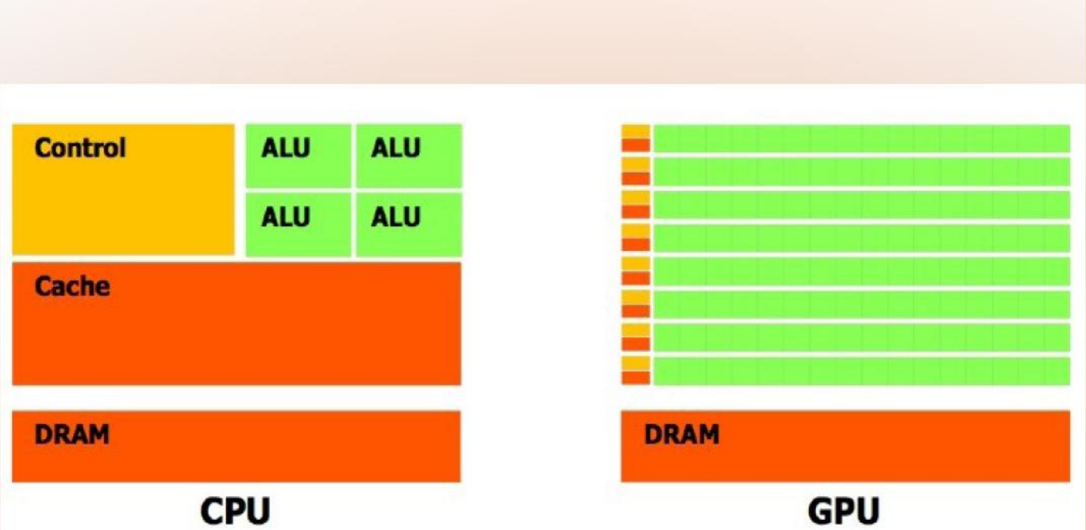
- Lets head back to our buddy Jim
- NVIDIA made the connection between scientific problems that need to be solved and the GPU
- In the world of Jim, this is analogous to Jim leaving his crowded workspace, walking down the hall and discovering an entire room full of hundreds of super geniuses!

CUDA & GPUS

- With the introduction of the programming language CUDA (short for Compute Unified Device Architecture), NVIDIA pushed the envelope of performance to never before seen heights
- By making use of CUDA, users can now harness the massive computational capabilities of the GPU and generate application specific code that can utilize the GPU (in addition to the CPU) for exceedingly complex problems
- For Jim, CUDA is a language that he can communicate with these super-geniuses with!

CUDA & GPUS

- Unlike the CPU, the GPUs primary job is to perform calculations
- The CPU has to worry about applications, the operating system and many other processes

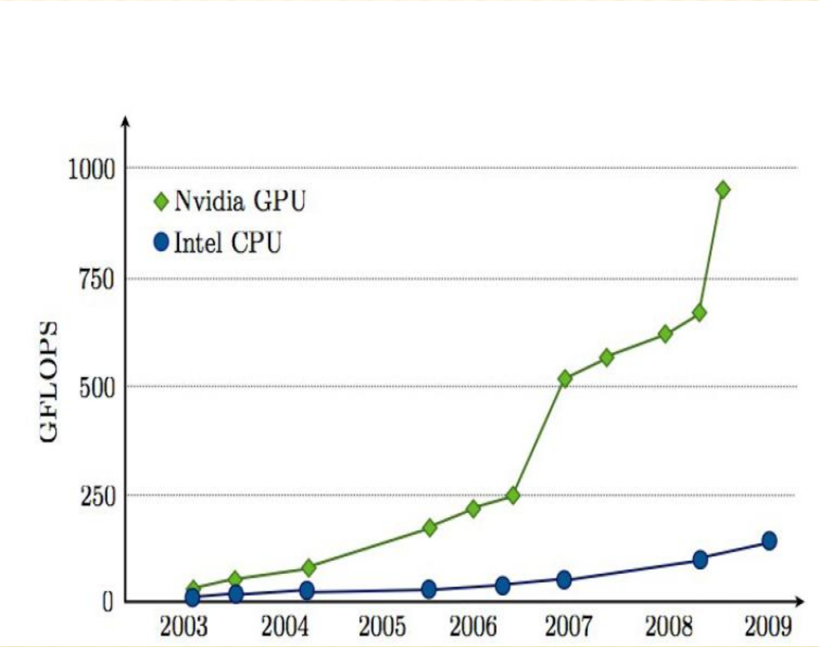


The GPU Devotes More Transistors to Data Processing

ALUs are the primary "calculators" in on the CPU and GPU

CUDA & GPUS

- Performance of GPUs and CPUs is often measured by a term: GFLOPS
- Giga (10^9) Floating Point Operations Per Second
- How many calculations it can do per second



CUDA & GPUS

- Previous image clearly shows that the GPU has always out performed the CPU from a purely computational perspective
- In order to harness the massive power of the GPU it must be programmed for your specific application
- Two main languages for this: CUDA and OpenCL
- OpenCL is much newer than CUDA and shows significant promise as a possible long term replacement for CUDA
- Both hardware and device independent

CUDA & GPUS

- Despite OpenCL's promise, CUDA is still the most popular GPU language out there and therefore will be focused on here
- C programming language derivative
- If you can program well in C, you can easily learn to use CUDA
- Many similarities to C, but several significant differences between the two

CUDA PROGRAMMING MODEL

- GPU is considered to be a co-processor along side the main CPU
- The CPU is referred to as the *host* while the GPU is referred to as the *device*
- Both devices have their own memory
- Data needs to be copied between the two in order for them to share information
- This copying of data is one of the biggest bottlenecks in GPU computing as it is very computationally expensive
- In terms of our friend Jim, this means that the hallway he walked down is extremely long

CUDA PROGRAMMING MODEL

- Nearly all CUDA programs have the same general flow

Define the kernel (function that is to run on the GPU) and the arguments that are to be passed to it

Allocate space on the GPU for all the data required to perform the computations you desire

Specify the block and grid sizes for the kernel

Copy the data from the host to the device

Execute the kernel

Copy the data from the device to the host

Free the allocated device memory

CUDA PROGRAMMING MODEL

- Defining kernels is syntactically very similar to function declarations in standard C
- GPU kernels are distinguished from host functions by the type specifier __global__

```
__global__ myFunction(int* arg A, int* argB,int N)
```

Kernels support both passing by value and passing by reference, just as in C

CUDA PROGRAMMING MODEL

- Memory allocation is one of the most subtle and intricate parts of GPU programming
- Performed by the function cudaMalloc() which takes in two arguments: address of a pointer to the object you wish to allocate and the size of the object you wish to allocate
- In laymen terms: cudaMalloc(whatImAllocating,howBig)

CUDA PROGRAMMING MODEL

- A thread is the single smallest unit of processing that can be scheduled by an operating system
- Think of threads as little minions that you are there to do your bidding
- More minions = better performance!

CUDA PROGRAMMING MODEL

- Each thread has its own unique ID (name)
- A thread block is a group of threads that work on shared data stored on the GPU. Each block has its own unique ID (name)
- Since there is a maximum number of threads that can exist in a block, it is advantageous to aggregate the thread blocks together to form a grid
- Threads within a single block can communicate with one another seamlessly through shared memory. Threads in different blocks cannot communicate via shared memory

CUDA PROGRAMMING MODEL

- Number of threads per block is usually around 512, although this is usually tuned higher or lower (almost always a multiple of 16) depending on the hardware and application
- Size of the grid is typically limited by hardware, and also needs to be tuned as well
- For good performance, grid size < 256

CUDA PROGRAMMING MODEL

- Copying data between host and device is very easy
- cudaMemcpy()
- Takes in four parameters: pointer to where you're copying it to, pointer to what you're copying, how much you're copying, and what type of transfer (host to device, device to host etc)
- Executing kernels is also straight forward:

```
myFunction<<gridsize,blocksize>>(arguments)
```

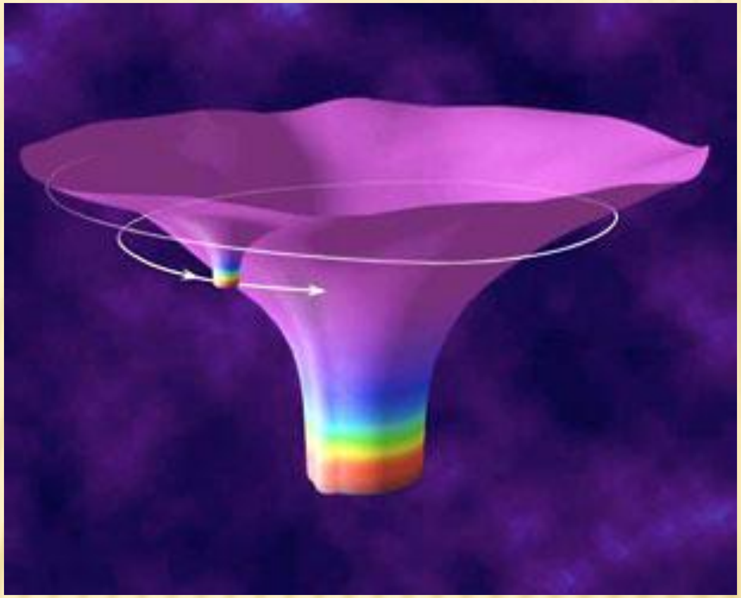
CUDA PROGRAMMING MODEL

- Enough of the gory details
- When the CPU is inefficient, it’s time to try the GPU
- Time for an example of what a GPU can do
- Apologize to any vegetarians in the room, but now it’s time for the real meat and potatoes of this talk

NUMERICAL RELATIVITY

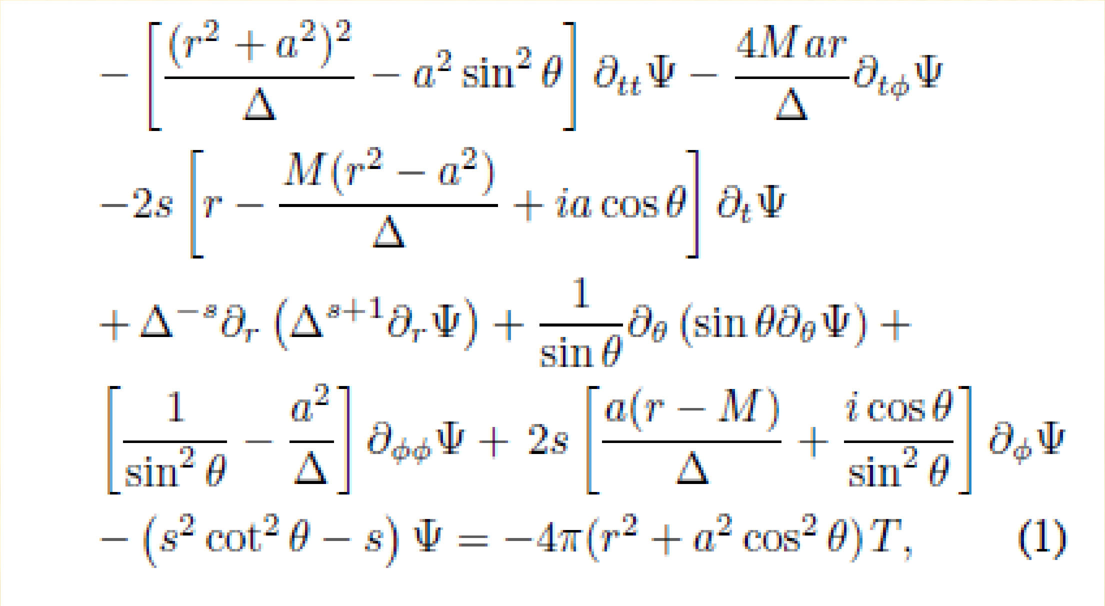
- Parts of the research work I've done is in an area of physics called Numerical Relativity
- Area of gravitational physics focused on the modeling of strong sources of gravitational waves
- These waves occur when massive objects (like really big – stars, planets, etc.) are accelerated out in space
- These waves have been predicted by Einstein's equations of relativity but have never been directly observed due to lack of technology (unavailable until now)
- The analysis of these waves allows for scientists to make new kinds of astronomical observations

APPLICATION EXAMPLE



- Extreme Mass Ratio Inspiral (EMRI)
- Evolves GWs generated by a compact object in a decaying orbit around a Super Massive Black Hole
- Evolution is modeled by the Teukolsky Equation

APPLICATION EXAMPLE

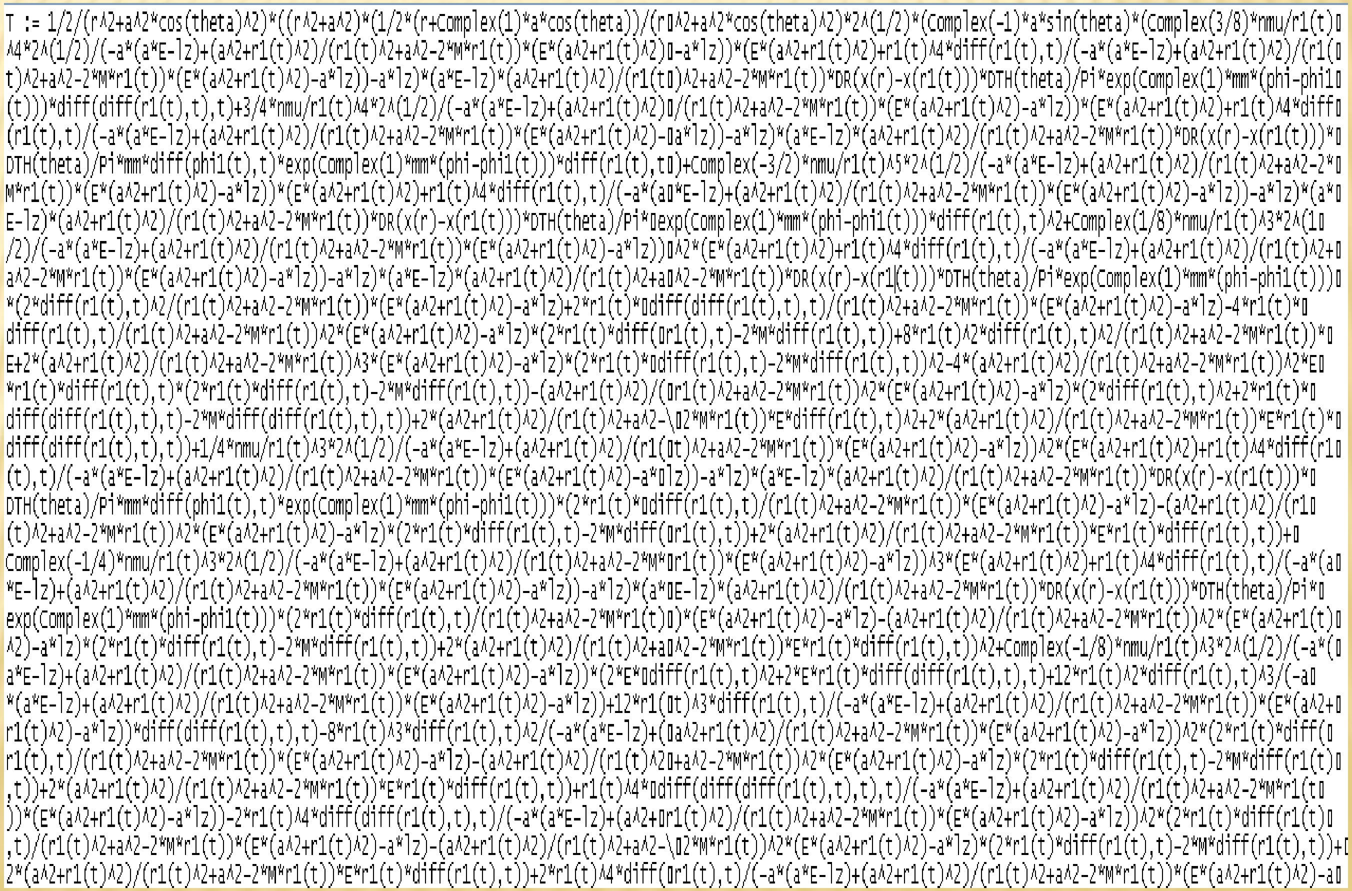


$$
\begin{aligned}
&-\left[\frac{(r^2+a^2)^2}{\Delta}-a^2\sin^2\theta\right]\partial_{tt}\Psi-\frac{4Mar}{\Delta}\partial_{t\phi}\Psi\\
&-2s\left[r-\frac{M(r^2-a^2)}{\Delta}+ia\cos\theta\right]\partial_t\Psi\\
&+\Delta^{-s}\partial_r\left(\Delta^{s+1}\partial_r\Psi\right)+\frac{1}{\sin\theta}\partial_\theta\left(\sin\theta\partial_\theta\Psi\right)+\\
&\left[\frac{1}{\sin^2\theta}-\frac{a^2}{\Delta}\right]\partial_{\phi\phi}\Psi+2s\left[\frac{a(r-M)}{\Delta}+\frac{i\cos\theta}{\sin^2\theta}\right]\partial_\phi\Psi\\
&-\left(s^2\cot^2\theta-s\right)\Psi=-4\pi(r^2+a^2\cos^2\theta)T, \qquad (1)
\end{aligned}
$$

- Doesn’t look like a particularly simple equation to do by hand
- Early work involved with solving for the source term, the T in the above equation

APPLICATION EXAMPLE

- Put this equation into maple to obtain an expression for the source term to use in our calculations



```
T := 1/2/(r^2+a^2*cos(theta)^2)*((r^2+a^2)*(1/2*(r+Complex(1)*a*cos(theta))/(r□^2+a^2*cos(theta)^2)*2^(1/2)*(Complex(-1)*a*sin(theta)*(Complex(3/8)*nmu/r1(t)□
^4*2^(1/2)/(-a*(a*E-lz)+(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))*(E*(a^2+r1(t)^2)□-a*lz))*(E*(a^2+r1(t)^2)+r1(t)^4*diff(r1(t),t)/(-a*(a*E-lz)+(a^2+r1(t)^2)/(r1(□
t)^2+a^2-2*M*r1(t))*(E*(a^2+r1(t)^2)-a*lz))-a*lz)*(a*E-lz)*(a^2+r1(t)^2)/(r1(t□)^2+a^2-2*M*r1(t))*DR(x(r)-x(r1(t)))*DTH(theta)/Pi*exp(Complex(1)*mm*(phi-phi1□
(t)))*diff(diff(r1(t),t),t)+3/4*nmu/r1(t)^4*2^(1/2)/(-a*(a*E-lz)+(a^2+r1(t)^2)□/(r1(t)^2+a^2-2*M*r1(t))*(E*(a^2+r1(t)^2)-a*lz))*(E*(a^2+r1(t)^2)+r1(t)^4*diff□
(r1(t),t)/(-a*(a*E-lz)+(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))*(E*(a^2+r1(t)^2)-□a*lz))-a*lz)*(a*E-lz)*(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))*DR(x(r)-x(r1(t)))*□
DTH(theta)/Pi*mm*diff(phi1(t),t)*exp(Complex(1)*mm*(phi-phi1(t)))*diff(r1(t),t□)+Complex(-3/2)*nmu/r1(t)^5*2^(1/2)/(-a*(a*E-lz)+(a^2+r1(t)^2)/(r1(t)^2+a^2-2*□
M*r1(t))*(E*(a^2+r1(t)^2)-a*lz))*(E*(a^2+r1(t)^2)+r1(t)^4*diff(r1(t),t)/(-a*(a□*E-lz)+(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))*(E*(a^2+r1(t)^2)-a*lz))-a*lz)*(a*□
E-lz)*(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))*DR(x(r)-x(r1(t)))*DTH(theta)/Pi*□exp(Complex(1)*mm*(phi-phi1(t)))*diff(r1(t),t)^2+Complex(1/8)*nmu/r1(t)^3*2^(1□
/2)/(-a*(a*E-lz)+(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))*(E*(a^2+r1(t)^2)-a*lz))□^2*(E*(a^2+r1(t)^2)+r1(t)^4*diff(r1(t),t)/(-a*(a*E-lz)+(a^2+r1(t)^2)/(r1(t)^2+□
a^2-2*M*r1(t))*(E*(a^2+r1(t)^2)-a*lz))-a*lz)*(a*E-lz)*(a^2+r1(t)^2)/(r1(t)^2+a□^2-2*M*r1(t))*DR(x(r)-x(r1(t)))*DTH(theta)/Pi*exp(Complex(1)*mm*(phi-phi1(t)))□
*(2*diff(r1(t),t)^2/(r1(t)^2+a^2-2*M*r1(t))*(E*(a^2+r1(t)^2)-a*lz)+2*r1(t)*□diff(diff(r1(t),t),t)/(r1(t)^2+a^2-2*M*r1(t))*(E*(a^2+r1(t)^2)-a*lz)-4*r1(t)*□
diff(r1(t),t)/(r1(t)^2+a^2-2*M*r1(t))^2*(E*(a^2+r1(t)^2)-a*lz)*(2*r1(t)*diff(□r1(t),t)-2*M*diff(r1(t),t))+8*r1(t)^2*diff(r1(t),t)^2/(r1(t)^2+a^2-2*M*r1(t))*□
E+2*(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))^3*(E*(a^2+r1(t)^2)-a*lz)*(2*r1(t)*□diff(r1(t),t)-2*M*diff(r1(t),t))^2-4*(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))^2*E□
*r1(t)*diff(r1(t),t)*(2*r1(t)*diff(r1(t),t)-2*M*diff(r1(t),t))-(a^2+r1(t)^2)/(□r1(t)^2+a^2-2*M*r1(t))^2*(E*(a^2+r1(t)^2)-a*lz)*(2*diff(r1(t),t)^2+2*r1(t)*□
diff(diff(r1(t),t),t)-2*M*diff(diff(r1(t),t),t))+2*(a^2+r1(t)^2)/(r1(t)^2+a^2-\□2*M*r1(t))*E*diff(r1(t),t)^2+2*(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))*E*r1(t)*□
diff(diff(r1(t),t),t))+1/4*nmu/r1(t)^3*2^(1/2)/(-a*(a*E-lz)+(a^2+r1(t)^2)/(r1(□t)^2+a^2-2*M*r1(t))*(E*(a^2+r1(t)^2)-a*lz))^2*(E*(a^2+r1(t)^2)+r1(t)^4*diff(r1□
(t),t)/(-a*(a*E-lz)+(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))*(E*(a^2+r1(t)^2)-a*□lz))-a*lz)*(a*E-lz)*(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))*DR(x(r)-x(r1(t)))*□
DTH(theta)/Pi*mm*diff(phi1(t),t)*exp(Complex(1)*mm*(phi-phi1(t)))*(2*r1(t)*□diff(r1(t),t)/(r1(t)^2+a^2-2*M*r1(t))*(E*(a^2+r1(t)^2)-a*lz)-(a^2+r1(t)^2)/(r1□
(t)^2+a^2-2*M*r1(t))^2*(E*(a^2+r1(t)^2)-a*lz)*(2*r1(t)*diff(r1(t),t)-2*M*diff(□r1(t),t))+2*(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))*E*r1(t)*diff(r1(t),t))+□
Complex(-1/4)*nmu/r1(t)^3*2^(1/2)/(-a*(a*E-lz)+(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*□r1(t))*(E*(a^2+r1(t)^2)-a*lz))^3*(E*(a^2+r1(t)^2)+r1(t)^4*diff(r1(t),t)/(-a*(a□
*E-lz)+(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))*(E*(a^2+r1(t)^2)-a*lz))-a*lz)*(a*□E-lz)*(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))*DR(x(r)-x(r1(t)))*DTH(theta)/Pi*□
exp(Complex(1)*mm*(phi-phi1(t)))*(2*r1(t)*diff(r1(t),t)/(r1(t)^2+a^2-2*M*r1(t)□)*(E*(a^2+r1(t)^2)-a*lz)-(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))^2*(E*(a^2+r1(t)□
^2)-a*lz)*(2*r1(t)*diff(r1(t),t)-2*M*diff(r1(t),t))+2*(a^2+r1(t)^2)/(r1(t)^2+a□^2-2*M*r1(t))*E*r1(t)*diff(r1(t),t))^2+Complex(-1/8)*nmu/r1(t)^3*2^(1/2)/(-a*(□
a*E-lz)+(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))*(E*(a^2+r1(t)^2)-a*lz))*(2*E*□diff(r1(t),t)^2+2*E*r1(t)*diff(diff(r1(t),t),t)+12*r1(t)^2*diff(r1(t),t)^3/(-a□
*(a*E-lz)+(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))*(E*(a^2+r1(t)^2)-a*lz))+12*r1(□t)^3*diff(r1(t),t)/(-a*(a*E-lz)+(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))*(E*(a^2+□
r1(t)^2)-a*lz))*diff(diff(r1(t),t),t)-8*r1(t)^3*diff(r1(t),t)^2/(-a*(a*E-lz)+(□a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))*(E*(a^2+r1(t)^2)-a*lz))^2*(2*r1(t)*diff(□
r1(t),t)/(r1(t)^2+a^2-2*M*r1(t))*(E*(a^2+r1(t)^2)-a*lz)-(a^2+r1(t)^2)/(r1(t)^2□+a^2-2*M*r1(t))^2*(E*(a^2+r1(t)^2)-a*lz)*(2*r1(t)*diff(r1(t),t)-2*M*diff(r1(t)□
,t))+2*(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))*E*r1(t)*diff(r1(t),t))+r1(t)^4*□diff(diff(diff(r1(t),t),t),t)/(-a*(a*E-lz)+(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t□
))*(E*(a^2+r1(t)^2)-a*lz))-2*r1(t)^4*diff(diff(r1(t),t),t)/(-a*(a*E-lz)+(a^2+□r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))*(E*(a^2+r1(t)^2)-a*lz))^2*(2*r1(t)*diff(r1(t)□
,t)/(r1(t)^2+a^2-2*M*r1(t))*(E*(a^2+r1(t)^2)-a*lz)-(a^2+r1(t)^2)/(r1(t)^2+a^2-\□2*M*r1(t))^2*(E*(a^2+r1(t)^2)-a*lz)*(2*r1(t)*diff(r1(t),t)-2*M*diff(r1(t),t))+□
2*(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))*E*r1(t)*diff(r1(t),t))+2*r1(t)^4*diff(□r1(t),t)/(-a*(a*E-lz)+(a^2+r1(t)^2)/(r1(t)^2+a^2-2*M*r1(t))*(E*(a^2+r1(t)^2)-a□
```

APPLICATION EXAMPLE

- It would take roughly 10-12 slides to include all of the terms that make up the source term (over 4000)
- The complexity of this equation originally caused it to take up the majority of the computation time regarding this equation when run on the CPU
- In order to decrease the simulation time, we coded the source term in CUDA and ran it on the GPU

SOURCE TERM RESULTS

- A Tesla C1060 GPU was used to speed up this calculation. Looks like it did an okay job:

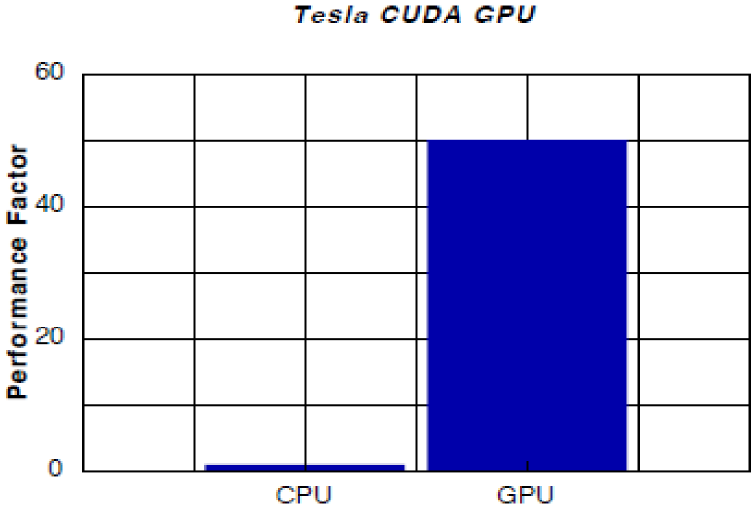
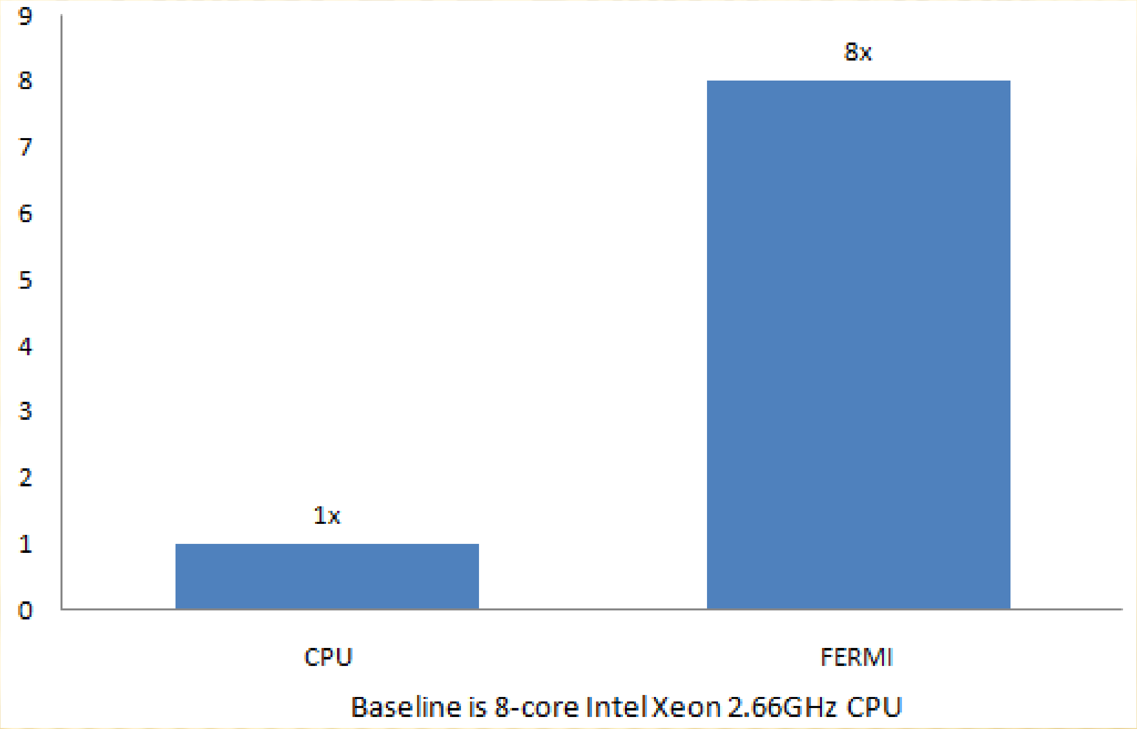


Figure 3. Overall performance of the EMRI Teukolsky code accelerated by the Tesla CUDA GPU. The baseline here is the supporting system's CPU – an AMD Phenom 2.5 GHz processor.

SOURCE TERM RESULTS

- The CPU we’re comparing performance with is no slouch - an AMD phenom quad-core 2.5GHz processor
- A 50x speed up over the CPU is extremely high

COMPLETE SIMULATION



- Based on these results, a single Fermi card is comparable to some x86 64 core CPUs!
- Significantly cheaper than a 64 core computer too!

GPU CHALLENGES

- Even with the significant performance gains obtained from utilizing the GPU, many people still do not use them
- Extracting any sort of near full-scale performance is no walk in the park
- Hundreds of thousands of CPU programming experts
- Very few GPU experts comparatively

GPU CHALLENGES

- Makes debugging complex errors quite difficult
- Often, people will run a simulation on a GPU and see no speed up and not understand why…
- Things like coalesced (sequential in memory) read and write operations, divergent branches (pieces of code that need to wait for other calculations to finish before continuing), local loads and stores (you have too much data for the GPU registers so it spills over into global memory) among many other issues

GPU CHALLENGES

- Very delicate and complicated (you get used to it after awhile) compared to standard CPU computing
- Explicit memory management
- Improper grid/block sizes
- Many, many ways to inhibit performance without meaning to
- Some applications just aren’t meant for the GPU – not parallelized, not complex enough (time it takes to transfer data and perform calculation is more than CPU takes to compute)

FUTURE OF GPUS

- AMD fusion architecture – CPU and GPU on the same chip – eliminates the GPU transfer time (instead of having Jim walk down a long hallway he simply walks across the hall)
- AMD's technology, if successful will ***significantly*** increase the value of GPU based work
- Many, many more applications will work efficiently on the GPU
- OpenCL and CUDA will become more efficient and easier to use
- Faster GPUs with more memory

MORE INFORMATION

- http://code.google.com/p/stanford-cs193g-sp2010/wiki/ClassSchedule
- http://drdobbs.com/
- http://forums.nvidia.com/index.php?act=ST&f=70&t=62620
- http://www.khronos.org/

REFERENCES

- http://thegadgetclub.net/wp-content/uploads/2010/11/nvidia-Quadro-4000-Mac.jpg
- http://demo.rockettheme.com/mar08_j15/
- http://www.gameguru.in/pc/2006/24/en7950gx2-2-pht-1g-dual-dual-graphics-card-launched-by-asus-india/
- http://www.khronos.org/