Evolving Use of GPU for Dassault Systemes Simulation Products

Luis Crivelli
And
Matt Dunbar

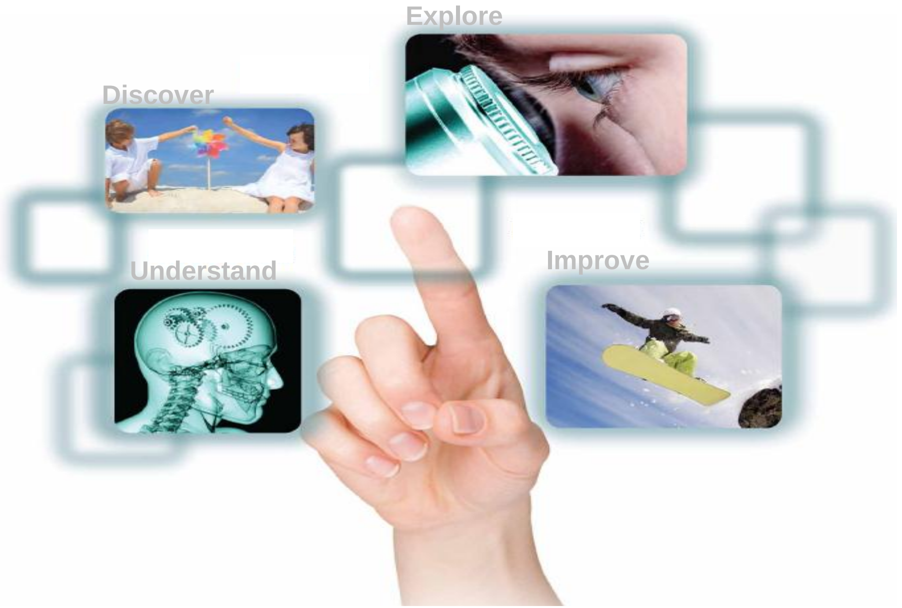
Dassault Systémes is dedicated to making…

Realistic Simulation

an integral
business practice

to Explore,
Discover,
Understand,
Improve

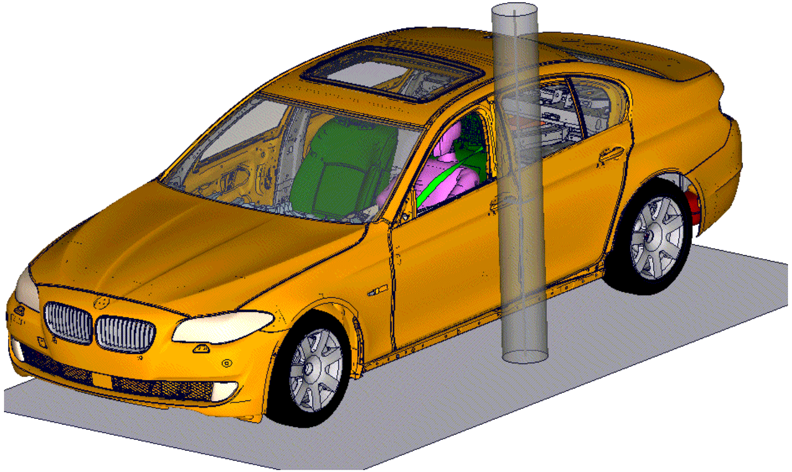
product, life, & nature



Simulation with Abaqus

Finite Element Simulation

Abaqus/Standard – static structural simulations
Abaqus/Explicit – short term dynamic simulations

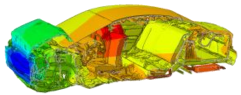


"Predictive Crashworthiness Simulation in a Virtual Design Process without Hardware Testing", Jurgen Lescheticky, Hariaokto Hooputra and Doris Ruckdeschel, BMW Group, SIMULIA Customer Conference, May 2010

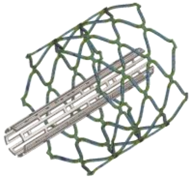
Simulation runtimes

- Simulation is a valuable part of an engineering design process, but computational cost is significant

Bottle Stacking
- Routine static analysis
- Compute time of 4-6 hours on 4 x86 cores



Automotive Crash
- Complex dynamics model
- Compute time of 2-5 days using between 16 and 32 x86 cores

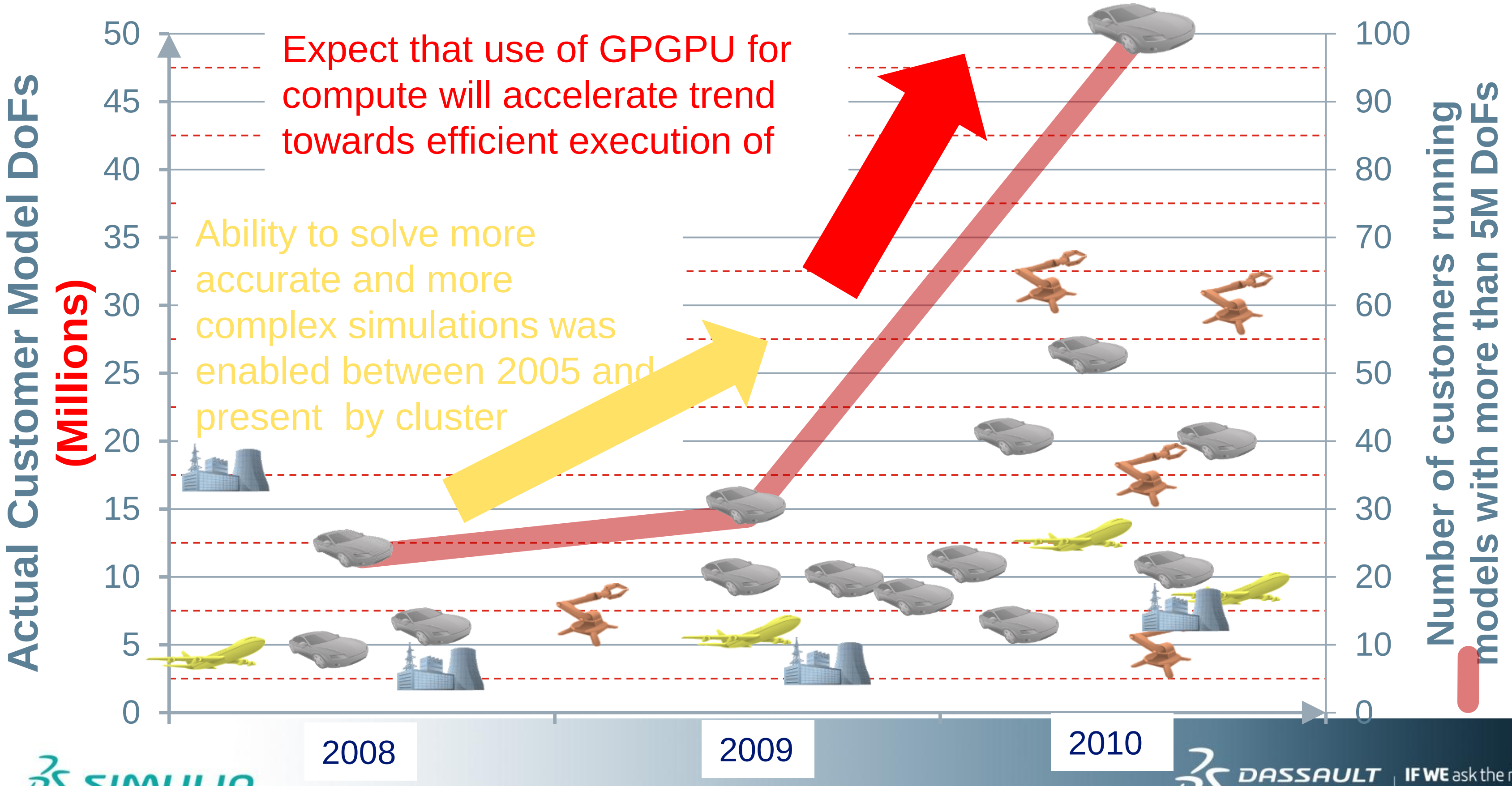


Stent Expansion
- Complex static analysis
- Compute time of 12-24 hours using between 8 and 32 x86 cores

Gasket Sealing
- Large static model
- Compute time of 2-5 days using between 32 and 64 x86 cores

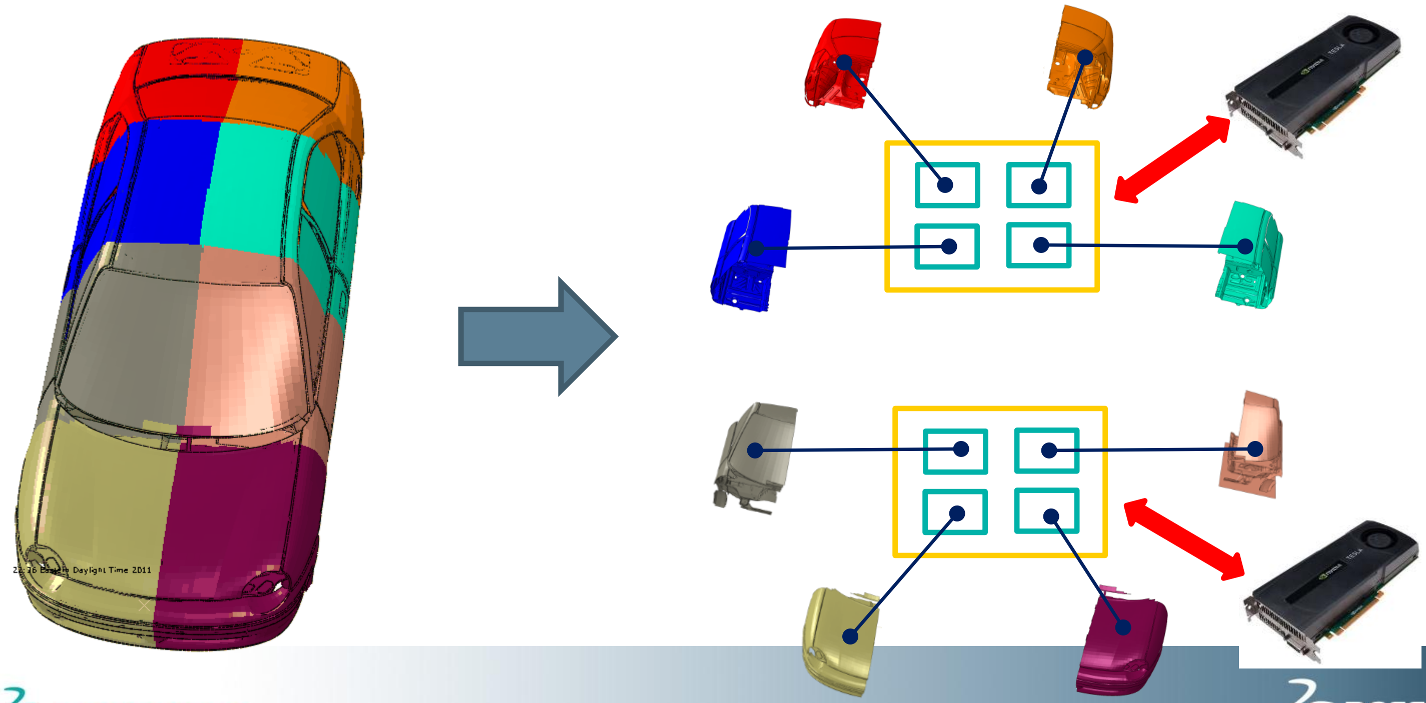
Reducing compute cost is critical!

Demand for Higher Accuracy



Simulation compute architecture

- Computational work is distributed to many cores on multiple servers by splitting a model into domains which are assigned to cores to parallelize computations

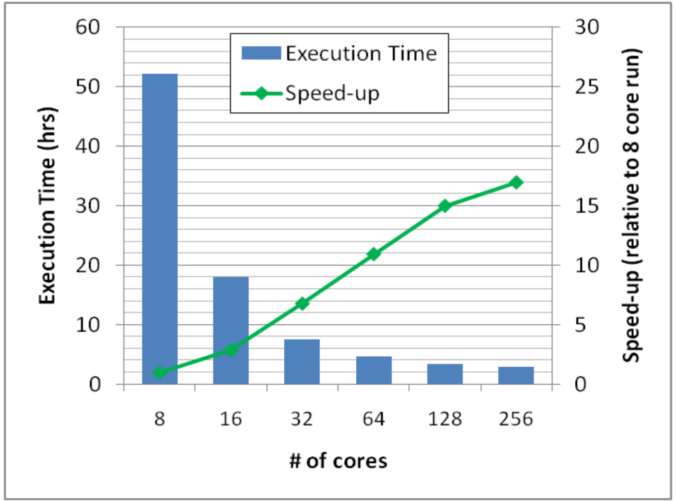


New code allows x86 cores to offload key compute work onto one or more Nvidia Tesla cards to accelerate computation

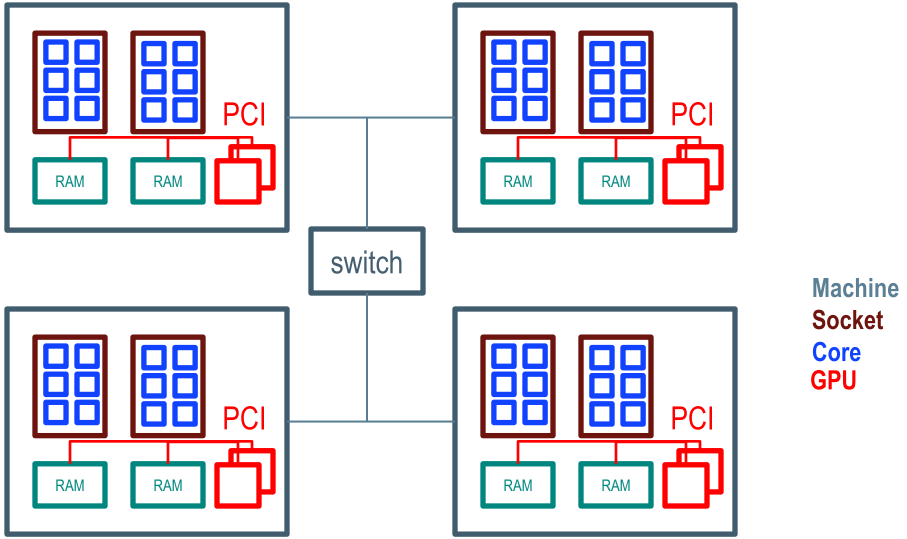
Abaqus

Existing Cluster-based Architecture

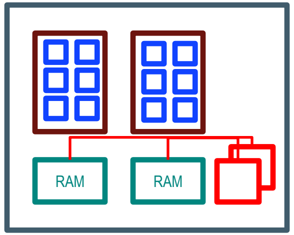
Abaqus applications allow users to exploit X86 clusters to decrease runtimes



Can GPU deliver a faster or more efficient solution?

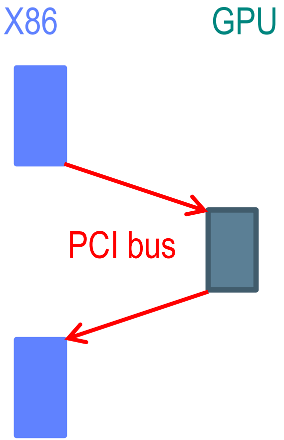


Approaches to Exploiting GPU



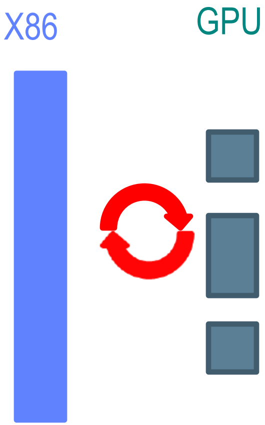
Offload Code to Card

- X86 cores are idle while GPU card processes



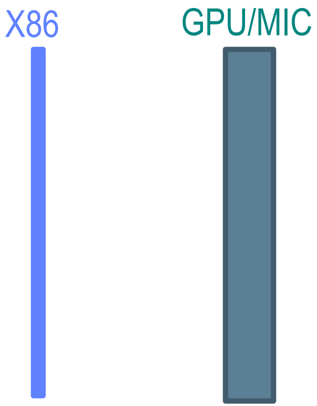
Hybrid Mode

- X86 cores and GPU card are used simultaneously with X86 assigning appropriate work to GPU



GPU As Platform

- Very limited “control” compute on X86; bulk of work done on GPU card



Abaqus Solvers

Key Code Components

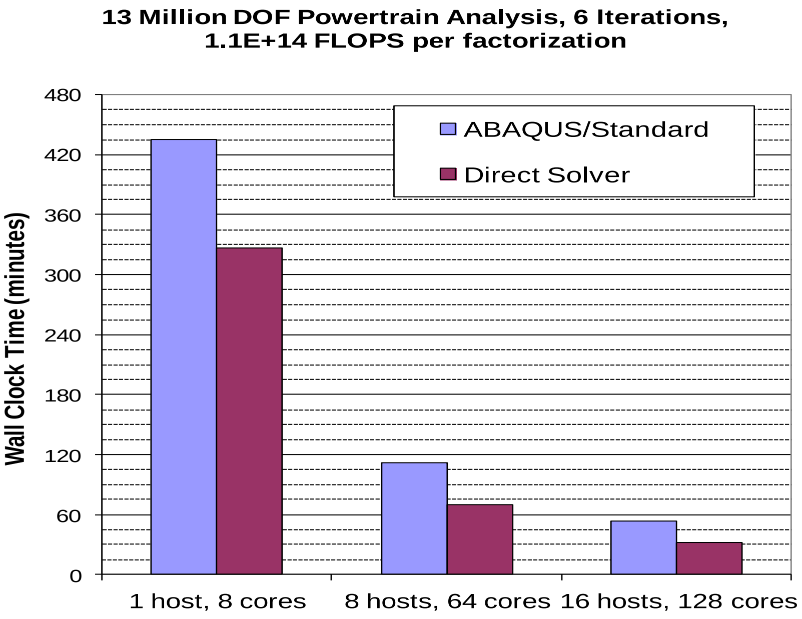
	Code Component	Cost of Code	Nature of Code
Abaqus/Standard	Linear Equation Solver	Increases with problem size.	Dense linear algebra kernels and "control code." Relatively small amount of code with high computational cost.
	Finite Elements	Most significant cost other than equation solver.	Naturally parallel but code is not written to expose SIMD parallelism.
Abaqus/Explicit	Elements	Typically 50%-75% of cost.	Naturally parallel. Code is written for SIMD architectures.
	Constraints and other	Much of remaining cost.	Complex parallelism.

Uses GPU in Abaqus 6.11 and 6.12

Focus of prototyping work

Abaqus/Standard

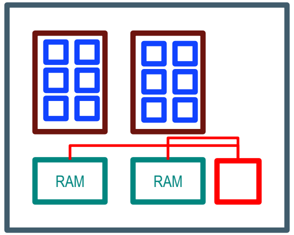
Time in Equation Solver

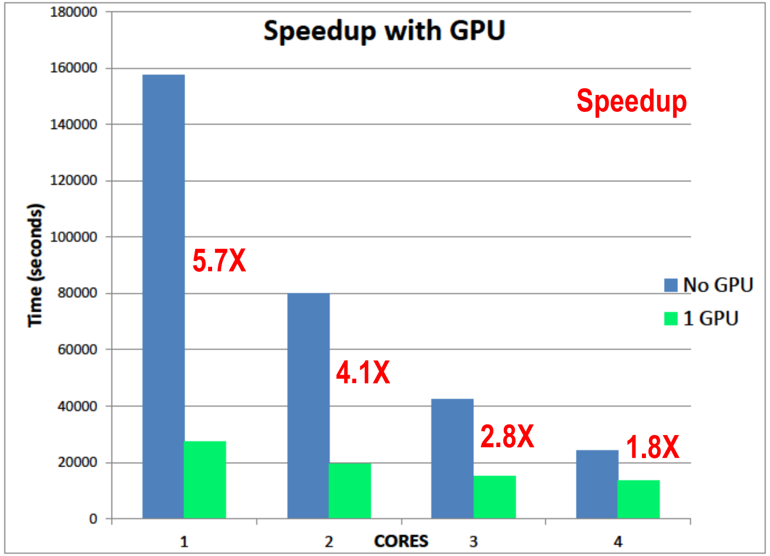


- As noted on previous slide Abaqus/Standard equation solver is a significant cost in execution of a simulation
- Natural target for GPU acceleration

Abaqus/Standard

Single GPU/Single Server

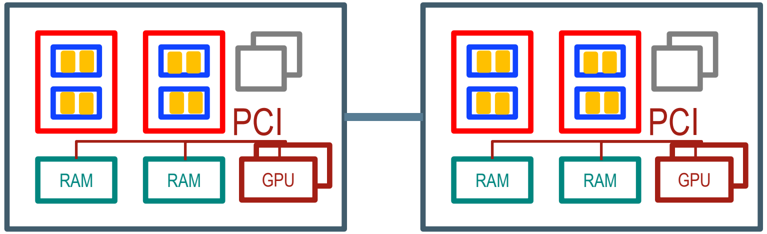


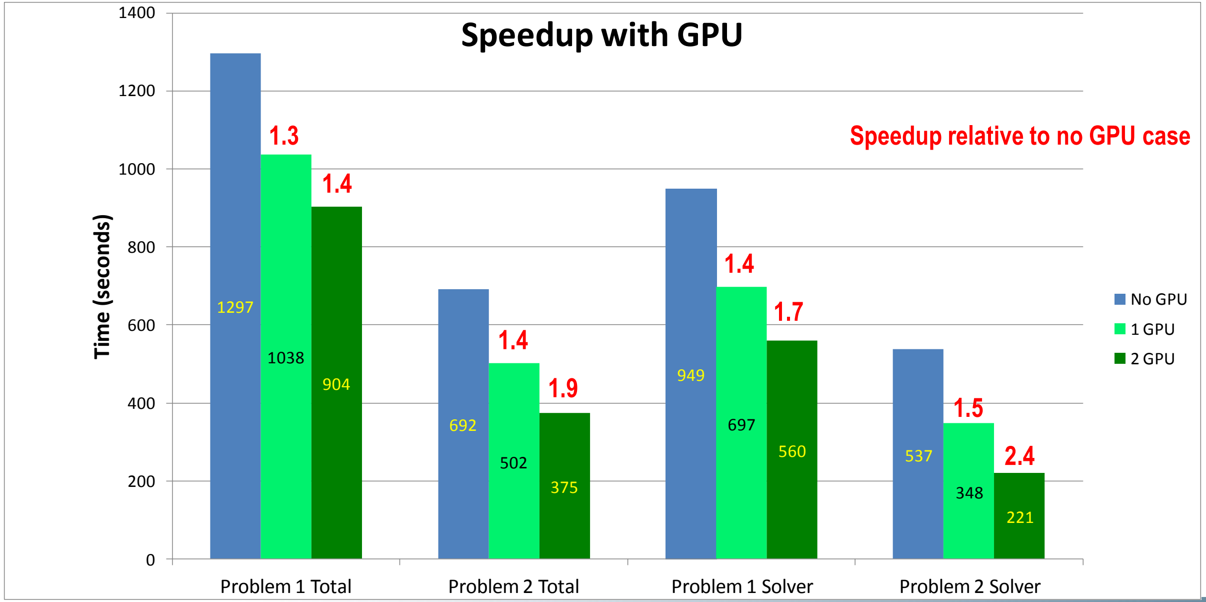


- At lower core counts GPU is more effective because the accelerated code takes a higher percentage of the overall time (code accelerated by the GPU is also effectively parallelized on X86 cores)
- Beyond 4 cores, X86 processors start overwhelming the GPU with computation

Abaqus/Standard

Cluster and multi-GPU support – hybrid mode





Jobs run on 2 hosts each with 12 cores, 48 GB of RAM, and 2 Nvidia GPU's

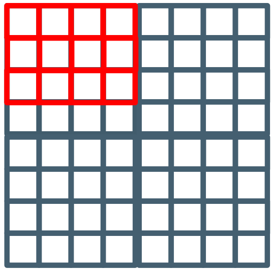
Problem 1
- 1.5 MDOF
- 5.37 teraflops per iteration

Problem 2
- 3.4 MDOF
- 10.8 teraflops per iteration

Abaqus/Explicit

GPU as a platform using Portland Group (OpenACC) compiler

- Explicit finite element codes do not have a natural bottleneck
  - Compute cost is spread through many 100's of routines
  - Code does have a natural SIMD structure
- SIMULIA has done a prototype in which the Portland Group compiler was used to build existing X86 code for GPU



Process group 2

Loop over groups of elements

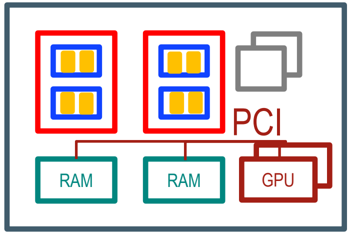
Loop A – process 1 to ngroup elements

Loop B – process 1 to ngroup elements

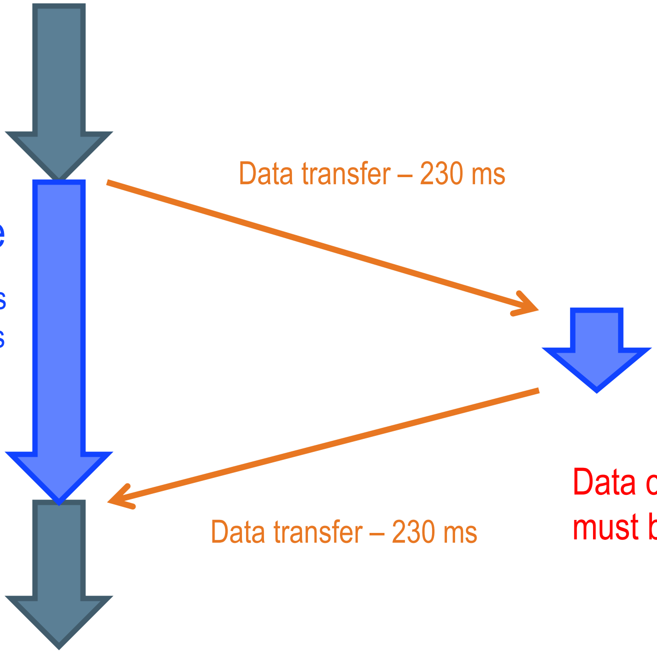
Loop C – process 1 to ngroup elements

Parallelization is done at a fine-grained level. Generally relies on elements being processed being identical (Single Instruction Multiple Data)

Abaqus/Explicit Prototype



X86 Socket | PCI bus | GPU



Data transfer – 230 ms

C3D8R Element Code

Xeon X5680 – 1 core – 357 ms
Xeon X5680 – 6 cores – 70 ms

Nvidia Tesla – 21 ms

Data transfer – 230 ms

Data cannot be transferred but must be left on GPU

Memory bandwidth bounds GPU performance

- Given peak performance of GPU near 1 TFlop vs peak performance for Westmere of ~70 Gflops why does Explicit code sped up by only 3X?

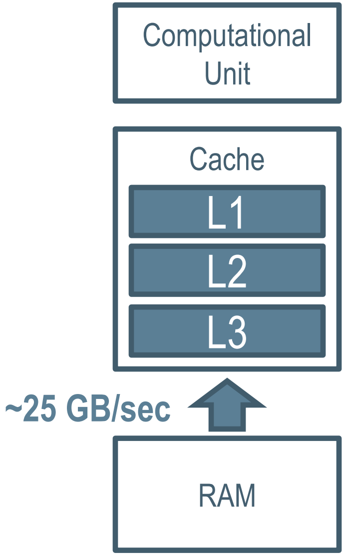
On X86 system temporary arrays remain in cache, but on GPU limited local memory is shared between a large number of threads so temporary data ends up be written back to system memory.

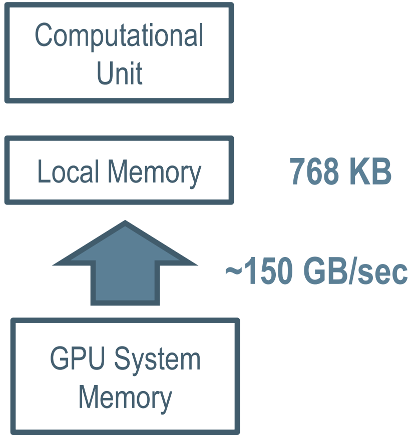
```
do k = 1, groupsize
   temp1(k) = in1(k) * in2(k)

do k = 1, groupsize
    temp2(k) = temp1(k) * in3(k)

do k = 1, groupsize
   out(k) = temp2(k) * x
```

High degree of data parallelism, but each piece of data is used a small number of times in operations





Conclusions

- Use of GPU as an X86 accelerator in a hybrid mode is effective and continued area of development

- Investigations into GPU as a platform are ongoing
  - Experience with OpenACC compiler approach has been good
  - Highly data parallel code does have bandwidth issues on GPU which limits gains
  - Data management between X86 memory and GPU is a key topic

IF WE™