2011 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
MODELING & SIMULATION, TESTING AND VALIDATION (MSTV) MINI-SYMPOSIUM
AUGUST 9-11 DEARBORN, MICHIGAN

ENABLING COMPUTATIONAL DYNAMICS IN DISTRIBUTED COMPUTING ENVIRONMENTS USING A HETEROGENEOUS COMPUTING TEMPLATE

Dan Negrut
Toby Heyn
Andrew Seidl
Dan Melanz
Department of Mechanical Engineering
University of Wisconsin - Madison
Madison, WI

David Gorsich
David Lamb
US Army Tank Automotive Research, Development and Engineering Center (TARDEC)
Warren, MI

ABSTRACT

This paper describes a software infrastructure made up of tools and libraries designed to assist developers in implementing computational dynamics applications running on heterogeneous and distributed computing environments. Together, these tools and libraries compose a so called Heterogeneous Computing Template (HCT). The underlying theme of the solution approach embraced by HCT is that of partitioning the domain of interest into a number of sub-domains that are each managed by a separate core/accelerator (CPU/GPU) pair. The five components at the core of HCT, which ultimately enable the distributed/heterogeneous computing approach to large-scale dynamical system simulation, are as follows: (a) a method for the geometric domain decomposition; (b) methods for proximity computation or collision detection; (c) support for moving data within the heterogeneous hardware ecosystem to mirror the migration of simulation elements from subdomain to subdomain; (d) parallel numerical methods for solving the specific dynamics problem of interest; and (e) tools for performing visualization and post-processing in a distributed manner.

INTRODUCTION

Over the last five years it has become apparent that future increases in computational speed are not going to be fueled by advances in sequential computing technology. There are three main walls that the sequential computing model has hit [1]. Firstly, there is the power dissipation wall caused by the amount of energy that is dissipated per unit area by ever smaller transistors. On a per unit area basis, the amount of energy dissipated by a Pentium 4 processor comes slightly short of that associated with a nuclear power plant. Since the amount of power dissipated scales with the square of the clock frequency, steady further clock frequency increases, which in the past were responsible for most of the processing speed gains, are unlikely. Forced cooling solutions could increase absolute clock rates, but come at a prohibitively high price.

The second wall, that is, the memory wall, arose in sequential computing as a manifestation of the gap between processing power and memory access speed, a gap that grew wider over the last decade. A single powerful processor will likely become data starved, idling while information is moved back and forth between the chip and RAM over a bus typically clocked at 10 to 30 GB/s. Ever larger caches alleviate the problem yet technological and cost constraints associated with large caches can't reverse this trend. This aspect will most likely be addressed by a disruptive technology such as photonic integrated circuits that promise to provide a new solution to bandwidth demand for on/off-chip communications [2].

Thirdly, investments in Instruction Level Parallelism (ILP), which mainly draws on instruction pipelining, speculative execution, and branch prediction to speed up execution represent an avenue that has already exhausted its potential. Both processor and compiler designers capitalized on opportunities for improved performance in sequential computing, which came at no cost to the software developer.

Augmenting branch prediction and speculative execution beyond current instruction horizons comes at a power price that in many cases increases exponentially with horizon depth.

While the sequential computing model paradigm seems to have lost, at least temporarily, its momentum, increases in flop rate over the next decade are guaranteed by vigorous miniaturization rates. Moore's law is alive and well, with 22 nm CMOS manufacturing technology slated to be made available by Intel in 2011, 15 nm in 2013, 11 nm in 2015, and 8 nm in 2017 [3]. This packing of more transistors per unit area, which translates into having more cores per chip, stands to address in the immediate future demands for higher flop rates and larger memory sizes in Scientific Computing. The success stories of this trend are the recent chip designs, code name Fermi and Knights, released by NVIDIA and Intel, respectively. The former packs approximately three billion transistors to lead to co-processors with 448 cores. Knights Ferry, announced in mid 2010, packs 32 cores in what is predicted to be the first in a family of products that belong to Intel's Many Integrated Core (MIC) architecture vision and slated to deliver using 22 nm technology a 50 core Knights Corner co-processor in 2011.

The co-processing idea is the enabler of the heterogeneous computing concept advertised recently as the paradigm capable of delivering exascale flop rates by the end of the decade. In this framework, run-time management tasks in a Scientific Computing program, such as high level flow control or I/O operations, are executed on the CPU. Conversely, math intensive computations applied uniformly to a large amount of data are pushed down to the GPU for single instruction multiple data (SIMD) execution. This heterogeneous computing concept is supported by hardware configurations such as the one shown in Figure 1. This is a 24 GPU and 48 core CPU heterogeneous cluster with a schematic provided in Figure 2. The cluster runs 64 bit Windows HPC Server 2008, has 48 GB of RAM on each compute node, and draws on 5,760 Tesla C1060 GPU scalar processors.

Figure 1: CPU-GPU cluster of the Modeling, Simulation, and Visualization Consortium at the University of Wisconsin.

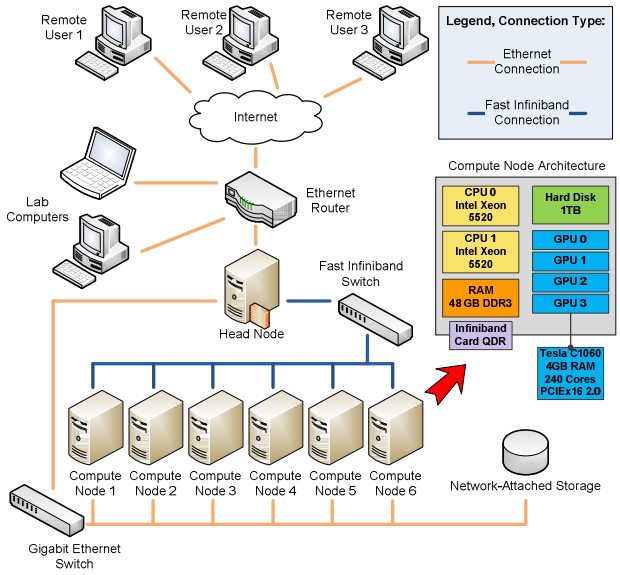

Figure 2: Schematic of CPU-GPU heterogeneous cluster.

A HETEROGENEOUS COMPUTATIONAL TEMPLATE FOR DYNAMICS PROBLEMS

From an abstract perspective, computational dynamics aims at determining how a system of mutually interacting elements changes in time. Heterogeneous computing becomes relevant when the number of interacting elements is very large. In this case, partitioning the dynamics simulation for distributed heterogeneous execution becomes attractive since it can lead to efficiency gains and/or increased problem size. At a space-time physical level, the space in which the system evolves is split into subdomains. At a virtual level, the simulation is split between available CPU cores, each of which manages one GPU accelerator. A one to one mapping between a spatial subdomain and a CPU core/GPU accelerator pair is established and maintained over the duration of the simulation leading to a spatial subdivision of the problem.

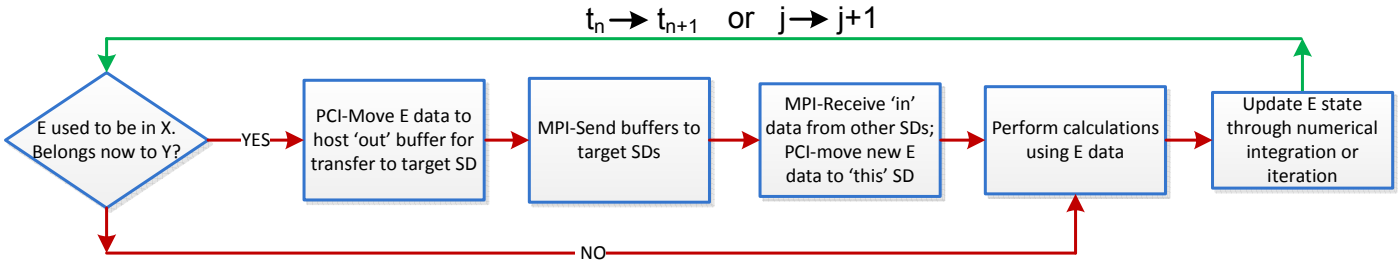

Figure 3: DDEP control flow. E stands for "element", SD stands for "sub-domain". Diagram captures what happens when advancing simulation to the next time step $t_n \rightarrow t_{n+1}$ or moving from iteration j to $j+1$.

At a minimum, high performance heterogeneous computational dynamics requires: (a) the ability to partition the problem according to the one-to-one mapping; i.e., spatial subdivision, discussed above (pre-processing); (b) a protocol for passing data between any two co-processors; (c) algorithms for element proximity computation; and (d) the ability to carry out post-processing in a distributed fashion. Requirement (a) is problem specific and is not discussed here. Requirement (d) is discussed in [4], which details how the visualization for post-processing has been sped up by more than one order of magnitude through the use of distributed computing and OptiX-enabled rendering on the GPU accelerators [5].

In terms of (b), we designed and implemented an early version of a Dynamic Data Exchange Protocol (DDEP) that manages transparently data passing between any two co-processors. Inter co-processor data exchange is mandatory in *any* distributed dynamics simulation that relies on spatial subdivision since elements leave one subdomain, which is virtually mapped to a certain co-processor, to enter a different subdomain. The need for such a protocol, schematically captured in Figure 3, along with its role can be illustrated easily in the context of a specific computational dynamics problem, such as molecular dynamics, smoothed particle hydrodynamics, or granular material dynamics. For molecular dynamics simulation, imagine a large 3D domain in which the motion of the atoms of interest will be investigated. This domain is partitioned into subdomains; the dynamics of all atoms that are physically present in a subdomain X will be determined using a host (CPU) thread h_X, which in turn manages co-processor (accelerator) a_X. DDEP is said to be dynamic because at each time step t_n the position of each element in the simulation, each atom in this case, needs to be accounted for. If at time step t_n the element finds itself outside the bounds of subdomain X, a two step process ensues: first, based on a global mapping available to each host thread and associated accelerator, the new owner sub-domain Y is identified. Second, all the information that defines the state of the element, which is problem specific but in the case of molecular dynamics given by the 3D position and generalized velocity of the atom, is loaded into a buffer that is sent to h_Y and from there to a_Y. The h_X to h_Y communication relies on the Message Passing Interface (MPI) standard [6]; the a_X to h_X and subsequently h_Y to a_Y communication is managed through CUDA calls [7]. The next release of DDEP will leverage GPUDirect technology in the existing cluster setup, which is expected to enable a further 30% reduction [8] in simulation times due to more efficient a_X to a_Y information passing. Essentially, using Mellanox InfiniBand adapters in conjunction with NVIDIA GPUDirect technology on Windows HPC Server 2008 allows for a skipping of two CPU-side memory transactions. Since the CPU-side memory space is shared in a virtual fashion by the co-processor, the page-locked virtual memory used on the CPU for the a_X to h_X CUDA and subsequently h_X to h_Y MPI memory transactions can draw on the same CPU physical memory space. The same holds true on the h_Y receiving side, where the h_Y to a_Y memory transaction can

originate from the same CPU physical memory space that served as the destination to the h_X to h_Y MPI transaction.

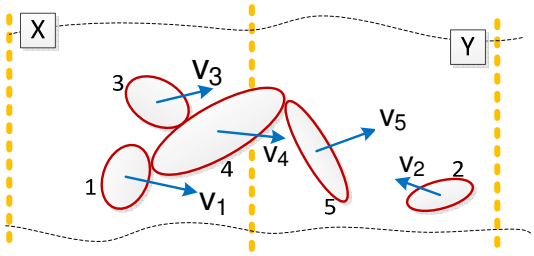

Figure 4: CPU thread h_X manages accelerator a_X that handles in this figure the time evolution of elements 1, 3, and 4. Although element 5 is in subdomain Y, its state information needs to be made available to subdomain X.

The decision of sending physical state information from a_X to a_Y in practical applications is complicated by the fact that X may need to have access to the state information of elements that are actually entirely contained within Y. In Molecular Dynamics, for instance, this is the case when a fast multipole algorithm is involved [9] due to long rage interactions; in smoothed particle hydrodynamics when a particle in Y falls within the compact kernel support of a particle contained in X; and in granular dynamics, when a body entirely contained within Y can collide with a body that although assigned to X spans the borders between X and Y. The latter scenario is captured in Figure 4, where body 5, which is entirely contained in the subdomain Y, collides with body 4 whose center of mass is contained in X. The process of marching in time (numerical time integration) is managed for bodies 4 and 5 by a_X and a_Y, respectively. Therefore, when a_X computes the forces acting on body 4 (needed by the numerical time integration algorithm), it will have to receive, using DDEP, the state information of body 5; i.e., its location and generalized velocity v_5. For Discrete Element Method approaches that use explicit numerical time integration [10], DDEP-enabled communication between X and Y occurs once per time step. For Differential Variational Inequality approaches state information should be exchanged using DDEP multiple times during the same time step due to a Cone Complementarity Problem that needs to be iteratively solved to recover the frictional contact force acting between bodies 4 and 5 [11]. Similarly, for approaches using implicit integration, communication must occur multiple times per time step during the iterative solution process. Such methods are feasible but may suffer from poor performance if the required communication load is too high.

The heterogeneous computing vision is abstracted into a general purpose Heterogeneous Computing Template (HCT) defined to streamline the implementation of any computational dynamics application (see Figure 5). In this framework, whose purpose is that of accelerating heterogeneous computing application development, DDEP provides the conduit between host/accelerator pairs; i.e., the h_X/a_X pairs. The data that needs to be transmitted using this conduit is physics specific and the user must provide the specific Partitioned Physics Services (PPS) required by the application. Proximity computations are always part of these services as seen for molecular dynamics, smoothed particle hydrodynamics, and granular dynamics applications. Applications in the latter two fields motivated our decision to concentrate on implementing efficient collision detection and nearest neighborhood approaches [12-14].

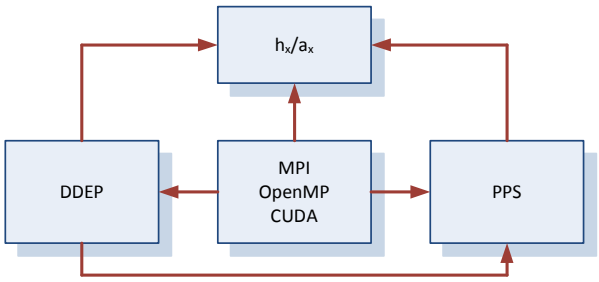

Figure 5: The proposed Heterogeneous Computing Template (HCT) is meant to streamline the development of computational dynamics applications that leverage CPU/GPU heterogeneous computing. Arrows from box A to box B indicate that box B relies on functionality/services provided by box A. Notation used: h_X/a_X, denotes the CPU/GPU pair handling computations associated with one problem subdomain; PPS represents the set of Partitioned Physics Services that encapsulate the physics of the specific computational dynamics problem considered. Both DDEP and PPS rely on the library support provided by MPI, OpenMP, and CUDA.

TEST CASES

A set of three examples are provided to illustrate how GPU computing and/or CPU/GPU computing have been used for large scale computational dynamics problems. The examples draw on many-body dynamics applications where the number of elements in the simulation can be in the millions. Experimental validation of these simulations is under way with early results reported in [10, 15].

Distributed Computing Using Multiple CPUs and the Message Passing Interface (MPI) Standard

The HCT concept has been used to implement a rigid-body dynamics simulation leveraging the Discrete Element Method (DEM). This implementation uses the Message Passing Interface (MPI) for communication between sub-domains. A description of the DEM approach, the MPI

implementation, and a numerical experiment are given in this sub-section.

The Discrete Element Method was proposed by Cundall to model the mechanical behavior of granular material [16]. The DEM can be classified as a penalty method, where the force acting between two colliding bodies is computed based on the associated interpenetration. In fact, the reaction force is often modeled by a spring-damper element where the spring and damping coefficients are determined from continuum mechanics theory or from experimental results [17]. At each time step, the DEM requires the following four computational steps: *i)* collision detection to determine pairs of colliding bodies; *ii)* computation of contact forces via constitutive relationship; *iii)* solution of Newton's Second Law to determine the accelerations of all bodies in the system; and *iv)* numerical integration to update velocities and positions.

Numerous contact force models have been developed and used over the years. For the purposes of this overview paper, a simple linear viscoelastic normal force model has been selected from the literature [18].

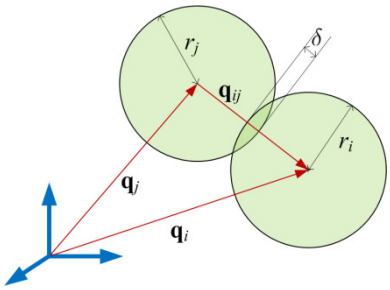

Figure 6: Schematic of DEM contact

Figure 6 shows two bodies i and j at positions $\mathbf{q}_i$ and $\mathbf{q}_j$ respectively. The spheres have radii r_i, r_j, and masses m_i, m_j. The normal force is modeled as a spring-damper acting at the interface between the two colliding bodies. The normal force is computed as follows [18]:

$$\mathbf{F}_n = k\,\delta\,\mathbf{n}_{ij} - \gamma\, m_{eff}\,\mathbf{v}_n \tag{1}$$

Here, δ is the normal compression, $\mathbf{n}_{ij}$ is the unit normal in the direction of $\mathbf{q}_{ij}$, $\mathbf{v}_n$ is the relative normal velocity, k is the spring stiffness, γ is the damping coefficient, and m_{eff} is the effective mass, $m_{eff} = m_i m_j \big/ \left(m_i + m_j\right)$. Note that the tangential force is neglected in this work for the sake of simplicity, but will be included in future work.

In this implementation, a pre-processing step discretizes the simulation domain into a specified number of sub-domains. The subdivision is based on a cubic lattice and is constant throughout the simulation. Separate MPI processes are mapped to each sub-domain. In this way, each process maintains a record of the bodies contained in the associated sub-domain and manages the solution of the DEM problem for those bodies.

Each body the simulation can have one of three states. First, it can be completely contained in a single sub-domain (*unique*). In this case, only one sub-domain needs to know the state of the body. The remaining two states occur when a body spans the boundary between two sub-domains. In this case, the sub-domain containing the center of mass of the body has the *master* copy of the body, while any other sub-domain containing part of the body has a *slave* copy. The MPI process managing a given sub-domain processes bodies of all states, computing the net force acting on each body in the sub-domain due to contacts which occur within the sub-domain.

Communication occurs twice during each time step. Mid-step communication occurs after the force computation. At this time, each sub-domain which shares a given body exchanges the net contact force it found for that body with all other participating sub-domains. At the end of the mid-step communication, all sub-domains which share a given body have the correct, full value of the net contact force acting on that body. Then, numerical integration occurs, followed by end-of-step communication. Here, the state data (position and velocity) of each *master* body is sent to all *slave* copies to explicitly synchronize the system. Then, each sub-domain is analyzed to determine if any bodies have moved to different sub-domains. If a body has completely left one sub-domain or entered a new sub-domain, its data is sent and received via MPI calls, then the body is added to or removed from the appropriate sub-domain.

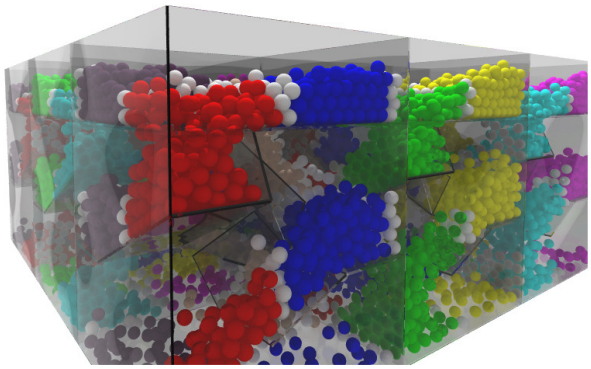

Figure 7: Snapshot of simulation with 36 sub-domains in a 6x6 grid. Bodies are colored by sub-domain, with shared bodies (those which span sub-domain boundaries) shown in white.

Two scenarios have been created to test the DEM implementation of the HCT concepts. First, a collection of spheres was dropped into a container with complex boundaries (see Figure 7). The simulation used 36 sub-

domains, contained 18,144 bodies, used a time step of 1e-5 seconds, and simulated 35 seconds of dynamics.

The second scenario demonstrates a wheeled vehicle operating on granular terrain. The vehicle represents the Mars Spirit Rover, and consists of six wheels connected to the chassis by revolute joints. All six wheels are given a constant angular velocity of π rad/sec. The terrain is composed of 50,000 spherical particles, and uses 9 sub-domains. The simulation used a time step of 1e-5 seconds and represents 15 seconds of dynamics. Figure 8 shows the state of the system at 3.2 seconds into the simulation.

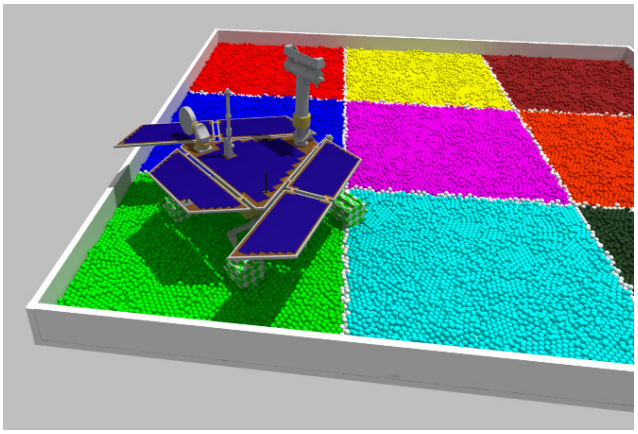

Figure 8: Snapshot of Mars Rover simulation with 50,000 terrain particles and 9 sub-domains.

In Figure 8, the wheels of the rover are checkered green and white to signify that the *master* copy of the rover is in the green sub-domain (lower-left corner) and the rover spans the sub-domains boundaries to the surrounding three sub-domains. Note that the rover is always considered as a unit when determining its sub-domain placement so that all of its data is together and its constraints can be analyzed correctly.

These two scenarios were simulated on the cluster described in the introduction, where each compute node has two Intel Xeon E5520 Quad core processors

Tracked-Vehicle Example

In the second test case, a model representing a light tracked vehicle was simulated operating on granular terrain. This simulation had a single domain and used a single CPU core/GPU accelerator pair. Additionally, this simulation used a different formulation of the multibody dynamics problem, leveraging the multibody dynamics engine Chrono::Engine [19]. Chrono::Engine uses a formulation based on a differential variational inequality (DVI). Compared to a penalty approach (as in DEM), the DVI approach enforces contacts as non-penetration constraints and allows larger integration time steps at the cost of higher computational effort at each step [11].

The model consisted of two tracks connected rigidly to a chassis (not shown in Figure 9). Each track contained five road wheels, three support rollers, one drive sprocket, one front idler, and 46 track shoes. The rollers were constrained to the chassis with revolute joints and the front idler was allowed to translate linearly in the plane of the track with a spring to maintain tension in the track. The collision geometry for this model was created via spherical decomposition [14]: the complex geometry of each component was approximated as a union of spheres so that fast parallel collision detection could be used. The sphere-set approximations of two of the track components can be seen in Figure 9. In total, the track model was represented using 3,189,816 spheres.

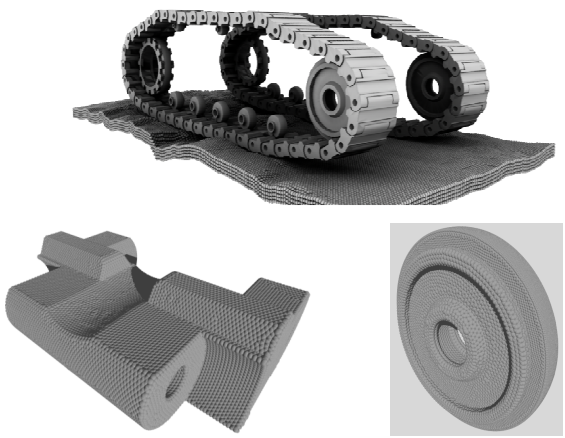

Figure 9: Snapshot from simulation of tracked vehicle on granular terrain (top), and examples of sphere-set collision geometry for two components of the track model (bottom).

The terrain model was created based on a height-map image. The model contained a total of 284,715 spheres in five layers. Each sphere had a radius of 0.027 m. The drive sprockets were both given a constant angular velocity of 1 rad/s. The 12 second long GPU simulation took 18.53 hours to finish with a time step of 0.01 seconds.

Note that the deformable nature of the terrain can be observed in Figure 10 by considering the vertical positions of the track shoes while on the bottom of the track. Given that the angular velocity of the driving gear was low, it was assumed (and qualitatively verified) that there was no slip in the driving direction. With no slip, the vertical position of a track shoe would be constant if the terrain was rigid, so any change in vertical position as the vehicle moves over the shoe can be attributed to sinkage in the granular terrain. This can be observed in Figure 10 where the positions of four consecutive track shoes attain a slight downward slope at about six seconds into the simulation, before each shoe is

picked up in turn by the drive sprocket to pass to the top span of the track.

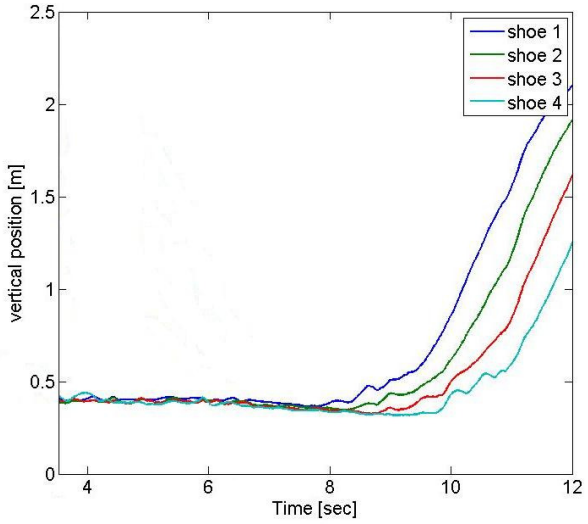

Figure 10: Positions of four track shoes during simulation on deformable granular terrain.

Collision Detection Example

Proximity computations are physics; i.e., problem, specific and represent one of the Partitioned Physics Services (PPS) that underlie the HCT approach. This third test case reports on a proximity computation, in the form of a collision detection task, which will support granular dynamics and fluid-structure interaction problems in HTC. Note that this test investigates only the collision detection task; i.e., given the instantaneous positions and orientations of a collection of objects, determine pairs of contacting objects and the associated contact data (contact normal, penetration, etc.). A first set of numerical experiments gauged the efficiency of the parallel collision detection algorithm developed. The reference used was a *sequential* implementation from Bullet Physics Engine, an open source physics-based simulation engine [20]. The CPU used in this experiment (relevant for the Bullet results) was AMD Phenom II Black X4 940, a quad core 3.0 GHz processor that drew on 16 GB of RAM. The GPU used was NVIDIA's Tesla C1060. The operating system used was the 64-bit version of Windows 7. The test was meant to gauge the relative speedup gained with respect to the serial implementation. This test stopped when dealing with about six million contacts (see horizontal axis of Figure 11), when Bullet ran into memory management issues. The plot illustrates a 180 fold relative speedup when going from sequential Bullet to the in-house developed GPU-based parallel implementation.

In the tracked vehicle simulation example, the proximity computation problem assumes the form of a collision detection task between elements participating in the model. Therein, due to the discrete representation of the terrain, the collision detection task requires the identification of several million contacts at each simulation time step. The number of collisions in granular terrain leads to demanding collision detection jobs; on average, in a cubic meter of sand one can expect approximately six *billion* contacts to take place at any instance of time. Heterogeneous computing was demonstrated to handle such problems effectively. Using four CPU cores and leveraging the OpenMP standard, four GPUs were used simultaneously to resolve collision detection tasks as described in [12]; the software and hardware stack along with the scaling of the implementation are illustrated in Figure 12 top and bottom, respectively.

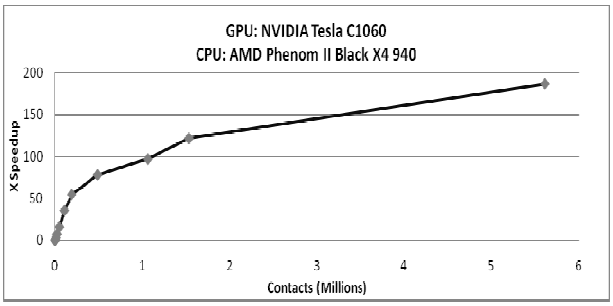

Figure 11: Overall speedup when comparing the CPU algorithm to the GPU algorithm. The maximum speedup achieved was approximately 180 times.

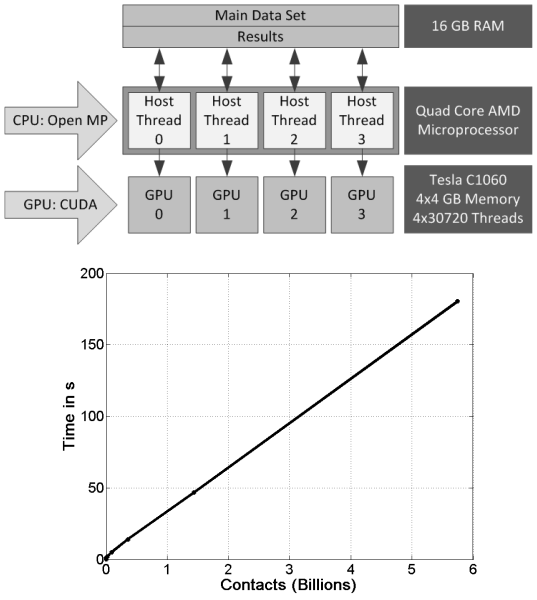

Figure 12: Software and hardware stack for collision detection algorithm (top). Computation time versus number of contacts, showing linear scaling up to six billion contacts.

DISCUSSION. WHAT COMES NEXT?

When compared to simulation results obtained using sequential computing, one of the examples considered in section 3 illustrated reductions in simulation times by a factor of 180. Recent granular dynamics results [11] indicate reduction in simulation times on the order of 50-60, and a factor of 60-70 for collision detection of ellipsoids [21]. Discrete Element Method simulation results reported by [10] suggest more than two orders of magnitude speedup when going from sequential to GPU-enabled parallel computing.

How are these speedups possible given that the difference in flop rate between GPU and CPU computing is, according to results in **Figure 13**, less than ten-fold? A first point that should be made is that the timing results report single (on the GPU) versus double precision (on the CPU) performance. Since for the latest Fermi generation of NVIDIA GPU cards [22], going from single to double precision arithmetic incurs a 2X slow down, this is an aspect that reduces any relative speedup by the same factor. Moreover, one might claim that aggressive tuning of the sequential code would significantly reduce sequential execution times, which has been shown to be the case in many applications.

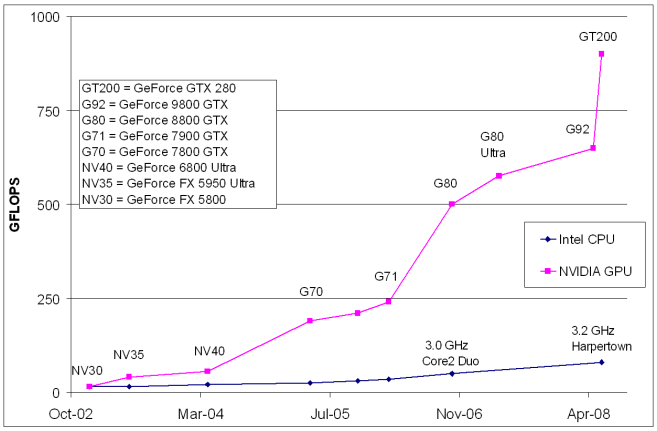

Figure 13: Evolution of flop rate, comparison CPU vs. GPU. GPU flop rates are reported in single precision, while CPU rates are for double precision arithmetic. For the latest generation of GPU cards Fermi [22], going from single to double precision arithmetic incurs a 2X slow down.

In regards to the first point, single versus double precision issues should be brought into the picture only for applications where double precision accuracy is crucial. In particular, for collision detection, validation tests carried out with hundreds of thousands of elements revealed no significant difference between the CPU double precision and GPU single precision results [12]. With respect to the second point, its lesson certainly applies for parallel executables as well: aggressive tuning of the parallel implementation can significantly reduce the parallel simulation times. If there is truly a point to be made it is that professional code, which most often assumes the form of libraries, should be used whenever possible, both for sequential and parallel programs. Yet this is easier said than done, since Scientific Computing, defined as the arena in which scientists and engineers use simulation to carry out novel research and solve challenging problems, is a niche where professional grade code has a relatively small foot print. This leads in many cases to ad-hoc implementations that see limited profiling and subsequent code optimization.

The efficiency gap between the sequential and parallel implementations can be traced back to several chip design features: hardware-implemented context switching, large memory bandwidth, and a relatively large number of registers per stream multiprocessor, which all place the GPU at an advantage. Of the factors mentioned, by far the most relevant is the fast context switching, which has to do with the ability of the scheduler to swap between warps of threads moving with virtually no overhead from a warp of threads that must wait for a memory transaction to a warp of threads that is ready for execution. As an example, for the Tesla C1060 NVIDIA model, the GPU has 30 stream multiprocessors (SMs), each with eight scalar multiprocessors (SPs). Each SM highly overcommits its resources, by managing for execution up to 1024 parallel threads, which are organized as 32 warps of 32 parallel execution threads each. The fast context switching and the SM's overcommitment decrease the likelihood of the eight SPs to idle. Therefore, when the execution code displays enough fine grain data parallelism, the fast context switching becomes an effective mechanism to hide high latency global memory accesses. This approach has permeated CPU designs in the form of the so called HTT, hyper-threading technology, where two execution threads are juggled by the CPU to hide memory latency. In addition to HTT, CPUs usually rely on large caches, albeit to a limited degree of success for applications where fine grain data parallelism can be exposed, a scenario where GPU computing excels.

The ability to hide memory latency with arithmetic operations through fast context switching, combined with a broad pipeline clocked at 140 GB/s that feeds the scalar processors on each stream multiprocessor are factors that motivated the Scientific Computing community to take a close look at heterogeneous computing as an alternative to traditional cluster-based supercomputing. The most aggressive adopter of this strategy is China, which over the last year deployed two of the fastest 10 supercomputers in the world. Both systems rely on GPU co-processing, one using AMD cards, the second, called Nebulae, using NVIDIA Tesla C2050 GPUs. In spite of a theoretical peak

capability of almost 3 petaflop/s, the highest in TOP500, Nebulae only holds the No. 2 position on the TOP500 list of the fastest supercomputers in the world. It clocked at 1.271 PFlop/s running the Linpack benchmark, which puts it behind Jaguar of Oak Ridge National Lab, which achieved 1.75 PFlop/s [23]. This apparent contradiction points out the current weakness of CPU-GPU heterogeneous systems: the CPU to/from GPU data transfer, typically an asynchronous process that can overlap computation on the accelerator, might represent a bottleneck that adversely impacts the overall performance of the system. Thus, in addition to overhead stemming from MPI communication over Infiniband Interconnect, or, for lesser systems, over Gigabit Ethernet, CPU-GPU heterogeneous systems require an additional layer of communication. It occurs over a PCI Express 2.0 interconnect clocked at 8 GB/s each way, and it represents a serious bottleneck that is partially alleviated by overlapping GPU execution and communication when using page lock memory transfers and virtual memory transactions. GPUDirect technology [8], briefly mentioned before, alleviates the transfer process through a protocol for the CPU-side memory management that stitches the virtual memory spaces associated with CPU-to-GPU and CPU-to-CPU data transfer. Yet it is clear that this represents a cumbersome and temporary solution that, given the potential of heterogeneous computing, will soon be replaced by hardware designs that eliminate the CPU-to-GPU PCI conduit. The most notable effort in this direction is AMD's Accelerated Processor Unit (APU), which through its Fusion concept aims at CPU and GPU same-die integration [24]. GPU access to a broader memory space that is physically shared by all die processors represents a side benefit of such integration. On top of the line GPUs, such as Fermi, this would extend the 4 GB of global memory that is often insufficient for keeping busy close to 500 scalar processors.

CONCLUSIONS

This paper provides an overview of the emerging paradigm of heterogeneous computing and summarizes a heterogeneous computing template (HCT) that promises to facilitate application development for large-scale computational dynamics problems. GPU computing opens up new avenues for the analysis at an unprecedented level of accuracy of large and challenging dynamic systems. This has been illustrated by examples that concentrate on problems with a large number of rigid bodies. Two order of magnitude reductions in simulation times and increases in problem size are demonstrated when using heterogeneous CPU/GPU computing for collision detection, where problems with up to six billion collision events were solved in less than three minutes. Although not discussed here, heterogeneous computing has motivated research into numerical methods for the parallel solution of large differential variational inequality problems [11], and has also been used very effectively in distributed visualization tasks where post-processing times for simulation visualization were reduced by more than one order of magnitude [4].

Beyond its immediate relevance in solving many-body dynamics problems, heterogeneous CPU-GPU computing promises to become a computational paradigm that can address stringent efficiency needs in Scientific Computing applications in diverse fields such as climate modeling, quantum chemistry, fluid dynamics, and biochemistry. This emerging computational paradigm is anticipated to become fully mature in the 2012-2013 timeframe, when the 22 nm technology is anticipated to enable same die CPU-GPU integration. To fully capitalize on the potential of heterogeneous computing, a focused and long term software design and implementation effort remains to address aspects specific to this new paradigm. In this context, OpenCL [25] represents an attempt at generating a programming framework for heterogeneous computing. However, it is not widely adopted and therefore a lack of software that draws on this standard and is aimed at computational dynamics problems continues to translate in ad-hoc implementations that fail to leverage the full potential of heterogeneous computing.

ACKNOWLEDGEMENTS

Financial support for this work was provided in part by a US Army grant. The first author was financially supported in part by the National Science Foundation grant NSF-CMMI-0840442. Dan Melanz and Makarand Datar are acknowledged for their editorial suggestions.

Disclaimer: Reference herein to any specific commercial company, product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the Department of the Army (DoA). The opinions of the authors expressed herein do not necessarily state or reflect those of the United States Government or the DoA, and shall not be used for advertising or product endorsement purposes.

REFERENCES

[1] Manferdelli, J.L., *The Many-Core Inflection Point for Mass Market Computer Systems.* CTWatch Qtrly, 2007. **3**(1).

[2] Chan, J., G. Hendry, A. Biberman, and K. Bergman, *Architectural Exploration of Chip-Scale Photonic Interconnection Network Designs Using Physical-Layer Analysis.* J. Lightwave Technol., 2010. **28**: p. 1305-1315.

[3] Skaugen, K., *Petascale to Exascale: Extending Intel's HPC Commitment: http://download.intel.com/pressroom/archive/reference/ISC_2010_Skaugen_keynote.pdf* in *International Supercomputer Conference*. 2010.

[4] Hill, J., H. Mazhar, and D. Negrut, *Using OptiX and Windows HPC Server 2008 for Fast Rendering on a GPU Cluster: Technical Report TR-2010-02*. 2010, Simulation-Based Engineering Laboratory, University of Wisconsin: Madison, WI.

[5] OptiX. *Interactive ray tracing on NVIDIA Quadro professional graphics solutions: http://www.nvidia.com/object/optix.html*. 2010 [cited 2010 August 27].

[6] Gropp, W., E. Lusk, and A. Skjellum, *Using MPI: portable parallel programming with the message-passing interface*. 1999: the MIT Press.

[7] NVIDIA. *Compute Unified Device Architecture Programming Guide 3.1: http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_CUDA_ProgrammingGuide_3.1.pdf*. 2010.

[8] GPUDirect. *Technical Brief - NVIDIA GPUDirect Techonology: Accelerating GPU-based Systems: http://www.mellanox.com/pdf/whitepapers/TB_GPU_Direct.pdf*. 2010 [cited 2010 August 30].

[9] Greengard, L. and V. Rokhlin, *A fast algorithm for particle simulations.* Journal of Computational Physics, 1987. **73**(2): p. 325-348.

[10] Tupy, M., *A Study on the Dynamics of Granular Material with a Comparison of DVI and DEM Approaches, M.S. Thesis*, in *Mechanical Engineering*. 2010, University of Wisconsin-Madison: Madison.

[11] Negrut, D., A. Tasora, M. Anitescu, H. Mazhar, T. Heyn, and A. Pazouki, *Solving Large Multi-Body Dynamics Problems on the GPU, book chapter* in *GPU Gems 4*, W. Hwu, Editor. 2010, Addison Wesley.

[12] Mazhar, H., T. Heyn, and D. Negrut, *A Scalable Parallel Method for Large Collision Detection Problems.* Multibody System Dynamics, 2011. **26**(1): p. 37-55.

[13] Hahn, P., *On the Use of Meshless Methods in Acoustic Simulations - M.S. Thesis*, in *Mechanical Engineering*. 2009, University of Wisconsin-Madison: Madison.

[14] Heyn, T., *Simulation of tracked vehicles on granular terrain leveraging GPU computing, M.S. Thesis*, in *Mechanical Engineering*. 2009, University of Wisconsin-Madison: Madison.

[15] Melanz, D., M. Tupy, B. Smith, T. Kevin, and D. Negrut, *On the Validation of a DVI Approach for the Dynamics of Granular Material*, in *ASME 2010 International Design Engineering Technical Conferences (IDETC) and Computers and Information in Engineering Conference (CIE)* 2010, American Society of Mechanical Engineers: Montreal, Canada.

[16] Cundall, P. and O. Strack, *A discrete element model for granular assemblies.* Geotechnique, 1979. **29**(1): p. 47-65.

[17] Johnson, K.L., *Contact Mechanics*. 1987, Cambridge: University Press.

[18] Silbert, L., D. Ertas, G. Grest, T. Halsey, D. Levine, and S. Plimpton, *Granular flow down an inclined plane: Bagnold scaling and rheology.* Physical Review E, 2001. **64**(5): p. 51302.

[19] Tasora, A. *Chrono::Engine - http://www.deltaknowledge.com/chronoengine/*. 2008.

[20] Erwin, C. *Physics Simulation Forum*. 2010 [cited 2010 January 15]; Available from: http://www.bulletphysics.com/Bullet/wordpress/.

[21] Pazouki, A., H. Mazhar, and D. Negrut, *Parallel Contact Detection between Ellipsoids with Applications in Granular Dynamics.* Mathematics and Computers in Simulation, Submitted, 2010.

[22] NVIDIA. *Fermi: Next Generation CUDA Architecture*. 2010 [cited 2010 January 30]; Available from: http://www.nvidia.com/object/fermi_architecture.html.

[23] TOP-500. *Top 500 Supercomputers June 2010: http://www.top500.org/lists/2010/06*. 2010 [cited 2010 August 31].

[24] AMD-Fusion. *The AMD Family of APUs: http://sites.amd.com/us/fusion/APU/Pages/fusion.aspx*. 2010 [cited 2010 August 30].

[25] Munshi, A., *The OpenCL Specification.* Khronos OpenCL Working Group, 2008.