Algorithms re-engineering as a fundamental step towards exploitable hybrid computing for engineering simulations

Francesco Iorio

Algorithmic Re-Engineering for Modern Non-Conventional Processing Units
September 30-October 2, 2009
CECAM-USI, Lugano, Switzerland

Autodesk Research
Agenda

- About Autodesk
- Simulation for Engineering trend in computing requirements
- Traditional approach to accelerators exploitation
- A more modern approach
- A new approach
Autodesk: a leading ISV in digital prototyping

Architecture, Engineering & Construction
- Architecture
- Civil & Structural Engineering
- Construction
- Mechanical, Electrical & Plumbing Systems
- Process Plant Design
- Real Estate

Automotive & Transportation
- Automotive
- Commercial & Recreational Transportation

Education
- Post-Secondary Education
- Secondary Education
- Students

Government
- Federal & National Agencies
- State & Local Government

Manufacturing
- Building & Construction
- Consumer Products
- Industrial Machinery
- Process Plants

Media & Entertainment
- Film
- Games
- Television

Utilities & Telecommunications
- Electric & Gas
- Telecommunications
- Water & Wastewater

Additional Solutions
- Geospatial
- Collaboration
Autodesk Research

- Research activities:
  - User interfaces
  - Environment & ergonomics
  - Simulation & graphics
  - High performance computing
  - Technology transfer

- High performance computing research group created in 2009
Simulation for Engineering trend in computing requirements

- Never-ending quest for efficiency and cost reduction
- Projects sustainability is becoming increasingly important
- Multidisciplinary, multidimensional analysis and simulations are required
- Design increasingly affected by simulation
Simulation for Engineering trend in computing requirements

1. From reduced complexity models to full models
2. From single-system to multi-system models
3. From individual simulations to multiple combined simulations
4. From simulation to optimization
Simulation for Engineering trend in computing requirements

- From reduced complexity models to full models
  - Simulation scalability issues
  - Overall simulation performance
  - Huge datasets directly from CAD and other sources
Simulation for Engineering trend in computing requirements

- From single-system to multi-system models
  - To reduce simulation time systems models are often reduced to small combinations of parts
  - The next barrier to increasing simulation accuracy involves modeling all the individual parts of a system and have them interact continuously in the simulation environment
Simulation for Engineering trend in computing requirements

- From individual simulations to multiple heterogeneous simulations
  - Simulations for the various dimensions/disciplines of a system are often conducted in isolation by individual experts
  - Only a combination of parameterized heterogeneous simulations can provide a comprehensive view on a project, essential to take informed decisions quickly and effectively
Simulation for Engineering trend in computing requirements

• From simulation to optimization
  – Simulation offers the analysis of a specific parameterized model
  – Increasing the efficiency and cost effectiveness of engineering projects involves finding the best combination of thousands of parameters
  – Faster and more precise simulations lead to faster convergence to optimal parameters
Requirements

• The size of future engineering simulation efforts requires enormous amount of computing power
• We need new methods for distributing simulations over numerous computing systems
• We need new methods for employing dedicated accelerators to reduce the simulations time and cost, and ultimately improve the overall projects efficiency
Example: Building Simulation and beyond

- Multiple simulation disciplines/dimensions:
  - Simulation for design
  - Simulation for structural integrity
  - Simulation for infrastructure
  - Simulation for green efficiency/lifecycle management
Example: Building Simulation and beyond

- So far these simulations have been conducted by domain experts in almost complete isolation, they need to be combined to achieve optimal results.
- From a single building to a single block, to a city, to a region, the situation just gets worse.
Example: Building Simulation for efficiency/lifecycle management

- Some example simulation components:
  - Ray tracing (sun exposure to maximize daylight exposure, heating and cooling issues)
  - CFD (HVAC to optimize air flow, temperature control and minimize related costs)
  - FEA (structural analysis, minimize bill of materials while ensuring tolerances)
  - Energy consumption/carbon footprint analysis

- Each of these simulations have both constraints and variables, which can be adjusted within certain boundaries
Example: Building Simulation for efficiency/lifecycle management

- The current state of the art is a set of separate simulations, with user-directed or brute-force choice of parameters to identify sets of “good” solutions.
- Optimization techniques can reduce the simulations iterations, but the computing requirements are still very large.
Simulation cost breakdown

- Simulation software development cost factors:
  - Code research and development cost: requirement to reduce time to market while exploiting performance improvement, software technology reuse
  - Code maintenance and porting cost: requirement to define “portable” algorithms, reducing the time to exploit new platforms and accelerators
Simulation cost breakdown

• Simulation software usage cost factors:
  – Required overall performance (time to answer)
  – Required overall precision (level of detail/optimization)
  – Software efficiency (tuning)
  – Hardware platform/s efficiency
  – Energy cost (running costs per simulation/optimization process)
How do non-conventional processing units and accelerators fit in all this?
Traditional approach to Accelerators exploitation

- So far accelerators have shown vast performance and efficiency improvements for extremely data parallel problems
- Non trivially-parallel algorithms still pose problems
- SaS business model emerging as new large market
- Hard to justify including accelerators into computing clouds/clusters for mainstream SaS usage due to development, maintenance and IT costs
Traditional approach to Accelerators exploitation

- Different architectural constraints require ad-hoc re-architecture tuning strategies
  - Programming languages/Instruction sets
  - Problem partitioning
  - Data layout
  - Memory hierarchy

- Parallel patterns and libraries contain solution to individual problems, but often their composition into applications often suffers badly from Amdahl’s law
Traditional approach to Accelerators exploitation

- OpenCL and similar languages are powerful tools, but do not scale to accommodate very large algorithms, especially on accelerators.
- Additional software infrastructure is often required to "glue" different parts of algorithms to avoid incurring in big performance penalties.
- Even stream-oriented languages don’t solve all problems: balancing heterogeneous software/hardware pipelines is hard to model, it is often a trial and error process.
Traditional approach to Accelerators exploitation

- Example engineering simulation scenario:

  Iterate thousands to millions of times
Traditional approach to Accelerators exploitation

- Example engineering simulation scenario:

![Diagram showing simulation steps and accelerators]
Traditional approach to Accelerators exploitation

• Simulations executed serially due to data dependencies
• Granularity is extremely coarse
• DGEMM et al. too coarse if present in large sequences
• Amdahl law hits hard at all synchronization points
• Intermediate resources (hosts) in the hierarchy are mostly unused and execute serial code or idle
• Difficult to model/predict all bottlenecks, especially due to the variability of computing systems
A more modern approach

Declarative language

+ Scalable task-based distributed software platform
A more modern approach

- Simulations are re-engineered into declarative, composable directed graphs; expressions of predefined libraries of small, computing intensive tasks (graph nodes) and data marshalling (graph edges)
- Task and data “core” libraries are written and optimized for CPUs and accelerators to provide implementations for the set of low-level tasks and data marshalling facilities
- Data dependency analysis is one of the fundamental techniques that enable optimizing compilers
A more modern approach

• Budimlic et al. [CPC09] show that splitting macro tasks (Cholesky factorization) to create finer-grained dependency graphs results in better usage of resources and better performance due to the different order of scheduling

• This shows the potential for emergent behavior in complex systems where hot-spots are difficult to predict

• Increased granularity improves portability reuse and interoperability between heterogeneous systems
A more modern approach

• Higher-level descriptive languages: some frameworks exist, some have visual editing environments, more to appear

• Large dependency graph analysis and integrated multi-level scheduling: a few frameworks exist (Intel Concurrent Collections, etc.), more to appear
A more modern approach, limitations

- Dependency graph partitioning, processing and distribution is not new, but the solution space for optimal distribution is too large to be treated with traditional heuristics, especially on heterogeneous systems.
- Complexity is much higher than classic “shop” problem: tasks do not execute in constant time even on similar platforms due to differences in memory architecture, cache sizes, number of cores, etc...
A more modern approach, limitations

- Task scheduling is lightweight and typically faster than context switch but not free
- Processing very large dependency graphs composed of hundreds of thousands of fine-grained tasks can bring even the best systems and task schedulers to their knees
- The overhead of scheduling the tasks could be so large to overshadow the benefits of parallelism
A new approach

- Declarative language
- Scalable task-based distributed software platform
- Dynamic compilation
- Iterative, feedback-directed task aggregation, scheduling and data transfer optimization
A new approach

• Dynamic compilation:
  – Dynamic, variable tasks granularity without “locking” algorithms implementations to a specific architecture
  – Reduce scheduling overhead by smart aggregation of tasks
  – Reduce tasks I/O overhead by collapsing data marshalling between tasks whenever possible
  – Additional potential for performance improvement due to further target-specific compilation optimizations
A new approach

• Platform specific dynamic compilation, task aggregation, data marshalling collapsing:
  – CPU: imperative language “core” tasks library, dynamic generation of object code kernels by LLVM dynamic compilation
  – Data parallel accelerator: OpenCL “core” tasks library, dynamic generation of kernels by dynamic OpenCL compilation, block size parameters
  – FPGA: VHDL/other languages “core” tasks library, dynamic generation of synthesizable kernels by dynamic compilation and synthesis
A new approach

• Iterative, feedback-directed task aggregation, scheduling and data transfer optimization:
  – Better chances of overcoming Amdahl’s law and exploiting emerging non conventional architectures by dynamically assigning tasks to the most appropriate computing resources while minimizing overhead
  – All computing resources are treated as such, thereby actively contributing to the graph evaluation (no host-accelerator relationship limitation)
A new approach

- **Dynamic compilation**: very few frameworks exist (LLVM, etc.), more to appear

- **Iterative, heterogeneous task assignment and scheduling optimization**: extremely few frameworks exist, *Autodesk Research* working on this
A new approach

- Our approach to Iterative, feedback-directed task aggregation, scheduling and data transfer optimization: Consider the scheduling optimization as a black-box non-linear constrained numerical optimization problem, use gradient-less function minimization techniques
A new approach

• Energy function examples: total wall clock, total used memory, total data communication, etc.
• Function parameters: tasks aggregation flags based on dependency constraints, tasks assignment flags based on available “core” tasks and data marshalling libraries
• Soft constraints examples: scheduling or assignments hints
• Hard constraints examples: accelerator/s memory usage, FPGA/s gates used
• Minimization techniques: GA, Simulated annealing, Pattern search
A new approach

• Fixed platform (embedded systems, games consoles, mobile phones, etc.): algorithms can be distributed in a pre-evaluated aggregation and scheduling configuration

• Variable platform (personal computers, etc.): algorithms are iteratively optimized at runtime on the target system
A new approach

• Recent, ongoing research project
• No measurable results to date
• We are writing a prototype based on Intel Concurrent Collections/TBB + OpenCL
• Testing will be conducted initially on ad-hoc synthetic benchmarks
Questions?
Thank you.