The Coming Era of Adaptive Control Systems in HPC

Laxmikant (Sanjay) Kale

http://charm.cs.illinois.edu
Just as I was preparing this

• I read an abstract of a talk yesterday:
  – “Supercomputing has had two "easy" decades”
    • where most of the increased performance of supercomputers
      came from the increase in uniprocessor performance
  • I thought we were having fun these decades
    – But not because it was easy
  • But then, I trust Marc Snir (who said this)...
    – And he did put those quotes
    – So, it means its going to get even harder
      • We all know why: sophisticated apps, complex machines
    – More fun, and more employment!
What *control systems* am I talking about?

- Runtime Systems?
- Java runtime:
  - JVM + Java class library
  - Implements JAVA API
- MPI runtime:
  - Implements MPI standard API
  - Mostly mechanisms
- I want to focus on runtimes that are “smart”
  - i.e. include strategies in addition to mechanisms
  - Many mechanisms to enable adaptive strategies
Why?

And what kind of adaptive runtime system I have in mind?

Let us take a detour
Governors

• Around 1788 AD, James Watt and Mathew Boulton solved a problem with their steam engine
  – They added a cruise control... well, RPM control
  – How to make the motor spin at the same constant speed
  – If it spins faster, the large masses move outwards
  – This moves a throttle valve so less steam is allowed in to push the prime mover
Feedback Control Systems Theory

• This was interesting:
  – You let the system “misbehave”, and use that misbehavior to correct it..
  – Of course, there is a time-lag here
  – Later Maxwell wrote a paper about this, giving impetus to the area of “control theory”

Control theory

• The control theory was concerned with stability, and related issues
  – Fixed delay makes for highly analyzable system with good math demonstration

• We will just take the basic diagram and two related notions:
  – Controllability
  – Observability
A modified system diagram

System

Output variables

Metrics that we care about

Observable/Actionable variables

Control variables

controller
Archimedes is supposed to have said, of the lever:

*Give me a place to stand on,
and I will move the Earth*

Need to have the lever

• Observability:
  – If we can’t observe it, can’t act on it

• Controllability:
  – If no appropriate control variable is available, we can’t control the system
    • (bending the definition a bit)

• So: **an effective control system needs to have a rich set of observable and controllable variables**
A modified system diagram

System

Output variables

Observable/Actionable variables

Control variables

controller

These include one or more:

- **Objective functions** (minimize, maximize, optimize)
- **Constraints**: “must be less than”, ..

Some of these are **Metrics** that we care about.
Feedback Control Systems in HPC?

• Let us consider two “systems”
  – And examine them for opportunities for feedback control

• A parallel “job”
  – A single application running in some partition

• A parallel machine
  – Running multiple jobs from a queue
A Single Job

• System output variables that we care about:
  – (Other than the job’s science output)
  – Execution time, energy, power, memory usage, ..
  – First two are objective functions
  – Next two are (typically) constraints
  – We will talk about other variables as well, later

• What are the observables?
  – Maybe message sizes, rates? Communication graphs?

• What are the control variables?
  – Very few. Maybe MPI buffer size? Bigpages?
Control System for a single job?

• Hard to do, mainly because of the paucity of control variables

• This was a problem with “Autopilot”, Dan Reed’s otherwise exemplary research project
  – Sensors, actuators and controllers could be defined, but the underlying system did not present opportunities

• We need to “open up” the single job to expose more controllable knobs
Alternatives

• Each job has its own ARTS control system, for sure
• But should this be:
  – Specially written for that application?
  – A common code base?
  – A framework or DSL that includes an ARTS?
• This is an open question, I think..
  – But it must be capable of interacting with the machine–level control system
• My opinion:
  – Common RTS, but specializable for each application
The Whole Parallel Machine

• Consists of nodes, job scheduler, resource allocator, job queue, ..

• Output variables:
  – Throughput, energy bill, energy per unit of work, power, availability, reliability, ..

• Again, very little control
  – About the only decision we make is which job to run next, and which nodes to give to it.
  – Maybe a few more ideas now, in the context of energy:
    • How many nodes to leave idle
    • What power limit to assign to a job
The Big Question/s:

How to add more control variables?
How to add more observables?

And then, how to build a powerful adaptive control system?
It so happens 😊

- My group’s research over the past 15–20 years can be thought of as a quest to add more observables and control variables
  - Programming models, languages, libraries, including:
    - Charm++, AMPI, Charisma, MSA, Charj,
- Now, I’d like to consolidate the experience and knowledge gained, and express it in a new *abstract programming model*
XMAPP

• XMAPP is an abstract programming model:
  – That means it characterizes a set of prog. models
• For a programming model to belong to this set, it must support
  – X: Overdecomposition
    • (as in: 8X objects than cores)
  – M: Migratability
  – A: Asynchrony
    • and Adaptivity, as a consequence of all the above
• So, XMAPP stands for:
  – Overdecomposition–based Migratability, Asynchrony and Adaptivity in Parallel Programming
Members of XMAPP–class

• The programming models in XMAPP, or exhibit some features of it
  – Charm++
  – Adaptive MPI
  – KAAPI
  – ProActive
  – FG–MPI (if it adds migration)
  – HPX (once it embraces migratability)
  – ParSEC
  – CnC
  – MSA (multi–phase Shared arrays)
  – Charisma
  – Charj
  – DRMS (old abstraction from IBM research..)
  – Chapel: may be a higher level model
  – X10: has asynchrony, but not migratable units
  – Tascel

Also, general work on adaptivity is relevant: Trilinos, Hank Hoffman/UIC, …
Over-decomposition

• Let the programmer decompose a computation into entities
  – Work units, data-units, composites
  – Into \textit{coarse-grained} set of objects
  – Independent of number of processors

• Let the entities communicate with each other without reference to processors
  – So each entity is like a virtual processor by itself

• Let an intelligent runtime system assign these entities to processors
  – RTS can change this assignment during execution

• This empowers the control system
  – A large number of observables
  – Many control variables created
Grainsize

• It is important to understand what I mean by coarse-grained entities
  – You don’t write sequential programs that some system will auto-decompose
  – You don’t write programs when there is one object for each float
  – You consciously choose a grainsize, BUT choose it independent of the number of processors
    • Or parameterize it, so you can tune later
Crack Propagation

This is 2D, circa 2002...
but shows over-decomposition for unstructured meshes..

Decomposition into 16 chunks (left) and 128 chunks, 8 for each PE (right). The middle area contains cohesive elements. Both decompositions obtained using Metis. Pictures: S. Breitenfeld, and P. Geubelle.
Grainsize example: NAMD

- High Performing examples: (objects are the work–data units in Charm++)
  - On Blue Waters, 100M atom simulation,
    - 128K cores (4K nodes), 5,510,202 objects
  - Edison, Apoa1 (92K atoms)
    - 4K cores, 33124 objects
  - Hopper, STMV, 1M atoms,
    - 15,360 cores, 430,612 objects
Grainsize: Weather Forecasting in BRAMS

- Brams: Brazilian weather code (based on RAMS)
- AMPI version (Eduardo Rodrigues, with Mendes, J. Panetta, ..)

Instead of using 64 work units on 64 cores, used 1024 on 64
Working definition of grainsize: amount of computation per remote interaction.

Choose grainsize to be just large enough to amortize the overhead.
Grainsize in a common setting

Jacobi3D running on JYC using 64 cores on 2 nodes

2048x2048x2048 (total problem size)

2 MB/chare, 256 objects per core
Impact on communication

- Current use of communication network:
  - Compute–communicate cycles in typical MPI apps
  - So, the network is used for a fraction of time,
  - and is on the critical path

- So, current communication networks are over-engineered for by necessity

BSP based application
Impact on communication

- With overdecomposition
  - Communication is spread over an iteration
  - Also, adaptive overlap of communication and computation

Overdecomposition enables overlap
Object-based over-decomposition: Charm++

- Multiple “indexed collections” of C++ objects
- Indices can be multi-dimensional and/or sparse
- Programmer expresses communication between objects – with no reference to processors
A[..].foo(…)

Scheduler

Message Queue

Processor 1

Scheduler

Message Queue

Processor 2
Note the control points created

• Scheduling (sequencing) of multiple method invocations waiting in scheduler’s queue
• Observed variables: execution time, object communication graph (who talks to whom)
• Migration of objects
  – System can move them to different processors at will, because..
• This is already very rich…
  – What can we do with that??
Optimizations Enabled/Enhanced by These New Control Variables

• Communication optimization
• Load balancing
• Meta-balancer
• Heterogeneous Load balancing
• Power/temperature/energy optimizations
• Resilience
• Shrink/Expand sets of nodes
• Application reconfiguration to add control points
• Adapting to memory capacity
Principle of Persistence

• Once the computation is expressed in terms of its natural (migratable) objects

• *Computational loads and communication patterns tend to persist, even in dynamic computations*

• So, recent past is a good predictor of near future

In spite of increase in irregularity and adaptivity, this principle still applies at exascale, and is our main friend.
Measurement-based Load Balancing

- Regular Timesteps
- Instrumented Timesteps
- Detailed, aggressive Load Balancing
- Refinement Load Balancing
XMAPP ideas and features have been demonstrated in full-scale production Charm++ applications
NAMD: Biomolecular simulations

• Collaboration with K. Schulten
• With over 45,000 registered users
• Scaled to most top US supercomputers
• In production use on supercomputers and clusters and desktops
• Gordon Bell award in 2002

Recent success:
Determination of the structure of HIV capsid by researchers including Prof Schulten
ChaNGa: Parallel Gravity

- Collaborative project (NSF)
  - with Tom Quinn, Univ. of Washington
- Gravity, gas dynamics
- Barnes–Hut tree codes
  - Oct tree is natural decomp
  - Geometry has better aspect ratios, so you “open” up fewer nodes
  - But is not used because it leads to bad load balance
  - Assumption: one-to-one map between sub-trees and PEs
  - Binary trees are considered better load balanced

Evolution of Universe and Galaxy Formation

With Charm++: use Oct–Tree, and let Charm++ map subtrees to processors
Spread of Infection:
Agent-based Simulation
An upcoming book
Surveys seven major applications developed using Charm++
Saving Cooling Energy

• Easy: increase A/C setting
  – But: some cores may get too hot
• So, reduce frequency if temperature is high
  – Independently for each core or chip
• But, this creates a load imbalance!
• No problem, we can handle that
  – Migrate objects away from the slowed-down procs
  – Balance load using an existing strategy
  – Strategies take speed of processors into account
• Implemented in experimental version
  – SC 2011 paper, IEEE TC paper
• Several new power/energy–related strategies
  – PASA ‘12: Exploiting differential sensitivities of code segments to frequency change
Fault Tolerance in Charm++/AMPI

• Four Approaches:
  – Disk-based checkpoint/restart
  – In-memory double checkpoint/restart
  – Proactive object migration
  – Message-logging with parallel restart: scalable fault tolerance

• Common Features:
  – Leverages object-migration capabilities
  – Based on dynamic runtime capabilities

• Several new results in the last year:
  – FTXS 2012: scalability of in-mem scheme
  – Hiding checkpoint overhead .. with semi-blocking..

Ships in Charm++ distribution, for years
Another idea for increasing controllable variables:

Reconfigurable Applications
App based Creation of Control Points

- A richer set of control points can be generated if we enlist help from the application
  - Or its DSL runtime, or compiler

- The idea is:
  - Application exposes some control knobs
  - Describes the effects of the knobs
  - The RTS observes performance variables, identifies the knobs that will help the most, and turns them in the right direction

- Examples: granularity, yield frequencies in inner loops, CPU–Accelerator balance
Load Balancing Framework

• Charm++ load balancing framework is an example of “customizable” RTS

• Which strategy to use, and how often to call it, can be decided for each application separately

• But if the programmer exposes one more control point, we can do more:
  – Control point: iteration boundary
  – User makes a call each iteration saying they can migrate at that point
  – Let us see what we can do: metabalancer
Meta-Balancer

• Automating load balancing related decision making

• Monitors the application continuously
  – Asynchronous collection of minimum statistics

• Identifies when to invoke load balancing for optimal performance based on
  – Predicted load behavior and guiding principles
  – Performance in recent past
Fractography: Without LB
Meta-Balancer on Fractography

- Identifies the need for frequent load balancing in the beginning
- Frequency of load balancing decreases as load becomes balanced
- Increases overall processor utilization and gives gain of 31%
Shrink/Expand job

• If a job is told to reduce the number of nodes it is using..
• It can do so now by migrating objects..
• Same with expanding the set of nodes used
• Empowered by migratability
Inefficient Utilization within a cluster

16 Processor system

Allocate A!

Conflict!

Job A

Job B

Current Job Schedulers can lead to low system utilization!
Adaptive Job Scheduler

- Scheduler can take advantage of the adaptivity of XMAPP jobs
- Improve system utilization and response time
- Scheduling decisions
  - Shrink existing jobs when a new job arrives
  - Expand jobs to use all processors when a job finishes
- Processor map sent to the job
  - Bit vector specifying which processors a job is allowed to use
    - 00011100 (use 3, 4 and 5!)
- Handles regular (non-adaptive) jobs
Two Adaptive Jobs

16 Processor system

Job A

Max_pe = 10
Min_pe = 1

Allocate B!

Allocate B!

B Finishes

Allocate A!

Job B

Min_pe = 8
Max_pe= 16
Rich Interaction desirable: currently there is very little
Whole machine runtime

• Job schedulers and resource allocators:
  – Accept more flexible QoS specifications from jobs
    • Creating more control variables
  – “moldable” specification:
    • This job needs between 3000–5000 nodes
    • Memory requirements...
    • Topology sensitivity, speedup profiles,...
  – Malleable:
    • this job can be told to shrink/expand after it has started
Whole machine control

• Monitor failures, and act in job-specific ways
• Global power constraints:
  – Inform, negotiate with and constrain jobs
• Thermal management
• I/O system and job I/O interactions
• Shrink and Expand jobs as needed to optimize multiple metrics
Novel, Revolutionary and Old?

• These concepts have been around for a while
  – E.g. Charm++ even in the present form is 13–15 years old

• An analogy might help
Dinosaurs, mammals and primates

• When the asteroid created a shock to the ecosystem
  – For us, multiple asteroids together:
    • End of frequency scaling,
    • Complex heterogeneous hardware,
    • Thermal, power, energy issues,
    • Component failures
    • Increasingly complex apps
  – Dinosaurs (well.. MPI) and mammals (XMAPP) both existed
    • But dinosaurs died out, mammals survived, and evolved further
    • The premium on “smart” rather than “big” in the ecosystem eventually saw the emergence of humans
      – Well.. Bending the truth a bit for the sake of analogy
    • Well, dinosaurs survived as birds… maybe MPI 5?
XMAPP models: adoption

• It is challenging to get the community to adopt a new programming model
  – And here we are talking about a whole class of them!

• It helps
  – To get a few from-scratch success stories
  – Some apps may get “refactored” to use the new model (Episimdemics)

• But large-scale adoption will be helped if we can support true “interoperability”
Interoperation of Parallel Languages

• Implement a library in the language that suits it the most, and use them together!

• MPI + UPC, MPI + OpenMP + Charm++

```c
int main(int argc, char **argv) {
    // Initialization
    mpi_module1(data);
}

mpi_module1(data) {
    // do work
    charm_module1(data);
}

charm_module1(data) {
    // do work
    charm_module2(data);
}

mpi_module2(data) {
    // do work
    mpi_module2(data);
}
```

(a) Time Division  
(b) Space Division  
(c) Hybrid
### Is Interoperation Feasible in Production Applications?

<table>
<thead>
<tr>
<th>Application</th>
<th>Library</th>
<th>Productivity</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARM in MPI (on Chombo)</td>
<td>HistSort in Charm++</td>
<td>195 lines removed</td>
<td>48x speed up in Sorting</td>
</tr>
<tr>
<td>EpiSimdemics</td>
<td>MPI IO</td>
<td>Write to single file</td>
<td>256x faster input</td>
</tr>
<tr>
<td>NAMD</td>
<td>FFTW</td>
<td>280 lines less</td>
<td>Similar performance</td>
</tr>
<tr>
<td>Charm++’s Load Balancing</td>
<td>ParMETIS</td>
<td>Parallel graph partitioning</td>
<td>Faster applications</td>
</tr>
</tbody>
</table>
Conclusions

• We need a much richer control system
  – For each parallel job
  – For parallel machine as a whole
• Current status: paucity of control variables
• Programming models can help create new observable and controllable variables
• As far as I can see,
  – XMAPP class programming models, with overdecomposition and migratability, and the resultant asynchrony and adaptivity are the main vehicle for this..
  – Do you see other ideas?
Conclusion

• HPC community suggestions:
  – Develop new XMAPP models
    • But: make sure you develop it in the context of at least two reasonable-size applications
  – Collaborate and compete on runtime adaptation strategies, based on the common assumptions of XMAPP models
    • Possibly develop standards for mature pieces

I am looking for a postdoc and/or a research programmer

See you at Charm++ BOF at SC: Tuesday noon

More info on Charm++: http://charm.cs.illinois.edu