Dynamic Load Balancing of High Performance Computing Applications

Echtzeit-AG, 25 Nov 2014, TU Dresden

Matthias Lieber (matthias.lieber@tu-dresden.de)
Center for Information Services and High Performance Computing (ZIH)
Technische Universität Dresden, Germany
Introduction

Dynamic Load Balancing
  – Objectives
  – Metrics: Workload, Load Balance
  – Typical Approach

Partitioning Methods

Software Stack

Experiences with COSMO-SPECS+FD4

Conclusion
Introduction: High Performance Computing

- Large number of computers (nodes) tightly coupled with fast network
- “Supercomputers”: fastest available HPC systems
- Batch scheduling of compute jobs
  - Applications request a fixed amount of nodes and time
- Typical programming model
  - Message Passing Interface (MPI)
  - Combined with OpenMP, OpenCL, CUDA, … within a node
- Current hot topics: energy efficiency, fault tolerance, heterogeneity, programmability

Tianhe-2, CN
- 16 000 nodes
- 384 000 cores
- + 48 000 Phi
- 54.9 PFLOPS
- 17.8 MW

Titan, USA
- 18 688 nodes
- 299 008 cores
- + 18 688 GPUs
- 27.1 PFLOPS
- 8.2 MW

Sequoia, USA
- 98 304 nodes
- 1 572 864 cores
- 20.1 PFLOPS
- 7.9 MW

K Computer, JP
- 88 128 nodes
- 705 024 cores
- 11.3 PFLOPS
- 12.6 MW

http://www.top500.org/featured/systems/titan-oak-ridge-national-laboratory/
http://www.top500.org/featured/systems/sequoia-lawrence-livermore-national-laboratory/
http://www.top500.org/featured/systems/kcomputer-kes2/
A few examples of HPC applications:

- Earth sciences: weather/climate prediction, earthquake simulations
- Structural mechanics: vehicle design, crash simulation, civil engineering
- Computational fluid dynamics: wind tunnel, turbine flow
- Molecular Dynamics: drug design, structural biology, material science

Many HPC applications are simulations based on partial differential equations.
Discretized in space and time to allow the approximate numerical solution.
Introduction: Discretization and Parallelization

- Grid represents distribution of unknowns in space
- Stencil computations to advance from one time step to the next
  - Data dependencies to neighbor cells only
Introduction: Discretization and Parallelization

- Grid represents distribution of unknowns in space
- Stencil computations to advance from one time step to the next
  - Data dependencies to neighbor cells only
- Parallelization by spatial decomposition of the grid (partitioning)
  - Load-balanced and minimal communication
Introduction: Discretization and Parallelization

- Grid represents distribution of unknowns in space
- Stencil computations to advance from one time step to the next
  - Data dependencies to neighbor cells only
- Parallelization by spatial decomposition of the grid (partitioning)
  - Load-balanced and minimal communication
Introduction: Unstructured Grids

- Rectangular grids are the most simple case
- Triangular meshes or arbitrary grid structures are also used
- Complex geometries are better represented

Institute of Aerospace Engineering, TU Dresden

Behrens, *Multilevel optimization by space-filling curves in adaptive atmospheric modeling*, Frontiers in Simulation, 2005
Few processes have more work (purple)
Most processes are waiting (red)

The colors on the process bars depict different activities: MPI sync and comm is red

64 Processes
Introduction: Sources of Imbalances

- Adaptive grids / Adaptive mesh refinement (AMR)
  - Adapt the spatial grid resolution dynamically to the simulation, e.g. shock waves, flame fronts, cracks, ...

- Adaptive time stepping
  - Same, but for time step size

Adaptive refinement of thermal plumes in the mantle convection simulation Rhea

Introduction: Sources of Imbalances

Model-inherent sources

- Computational effort per grid cell varies with the model variables
- Particle-in-Cell: number of particles per grid cell
- Cloud microphysics: presence of droplets, temperature

Laser wakefield acceleration simulation (LWFA) with particle-in-cell code PIConGPU

Cloud simulation COSMO-SPECS

Workload relative to avg

max: 91.7

max: 6.6
Outline

- Introduction

**Dynamic Load Balancing**
  - Objectives
    - Metrics: Workload, Load Balance
    - Typical Approach

- Partitioning Methods
- Software Stack
- Experiences with COSMO-SPECS+FD4
- Conclusion
Dynamic Load Balancing: Objectives

- Four objectives of dynamic load balancing
Four objectives of dynamic load balancing

- Balance workload

**BAD**
- load=13
- load=17

**GOOD**
- load=12
- load=12
- load=12
- load=12
Dynamic Load Balancing: Objectives

- Four objectives of dynamic load balancing
  - Balance workload
  - Reduce communication between partitions (due to data dependencies)
Four objectives of dynamic load balancing

- Balance workload
- Reduce communication between partitions (due to data dependencies)
- Reduce migration, i.e. communication when changing the partitioning
Four objectives of dynamic load balancing

- Balance workload
- Reduce communication between partitions (due to data dependencies)
- Reduce migration, i.e. communication when changing the partitioning
- Compute partitioning as fast as possible
Four objectives of dynamic load balancing

- Balance workload
- Reduce communication between partitions (due to data dependencies)
- Reduce migration, i.e. communication when changing the partitioning
- Compute partitioning as fast as possible

Contradictory goals

Optimal solution for first two goals is NP-complete

Existing methods (heuristics) provide different trade-offs between the four objectives
Outline

- Introduction
- **Dynamic Load Balancing**
  - Objectives
  - **Metrics: Workload, Load Balance**
  - Typical Approach
- Partitioning Methods
- Software Stack
- Experiences with COSMO-SPECS+FD4
- Conclusion
Dynamic Load Balancing: Metrics

- Workload / weight of a single grid cell
- Needs to be estimated for the future time step(s)
  - Typical: Measurement of current load (time, cycles, ...) and assume load will change slightly only (principle of persistence)
  - Derive suitable indicators from model-specific variables (i.e. number of particles in grid cell)


How to measure Load Balance?
How to measure Load Balance?

Focus on one time step
- i.e. one instance of the imbalance

Load Balance = \( \frac{\text{Avg. workload among procs}}{\text{Max. workload among procs}} \)

- Perfect balance: 1.0
- Worst case: \( \frac{1}{\text{number of procs}} \)
- Similar to utilization ratio of the processes
- Different definitions in the literature, but mostly all based on avg. and max. workload
Outline

- Introduction
- **Dynamic Load Balancing**
  - Objectives
  - Metrics: Workload, Load Balance
  - **Typical Approach**
- Partitioning Methods
- Software Stack
- Experiences with COSMO-SPECS+FD4
- Conclusion
Decompose the grid in *objects* for assignment to processes and migration between processes

- **Object** = Single grid cell or block of grid cells
- **Workload / weight** of a single object: $w_i$
Dynamic Load Balancing: Typical Approach

- Object size determines granularity
  - Too small objects: high overhead for management of objects and load balancing
  - Too large objects: too coarse grained to reach good load balance

- Estimation for required granularity when running on P processes
  \[ \max(w_i) \leq \frac{\sum w_i}{P} \]
  - To run efficiently on large number of processes: decrease \( \max(w_i) \) (i.e. object size) or increase \( \sum w_i \) (i.e. problem size) sufficiently

- Objects size may also influence cache efficiency of the computations
# Dynamic Load Balancing: Typical Approach

FOR timeStep = 1 TO numberOfTimeSteps

Determine load balance for this time step  
*(based on indicators or estimation from last time step)*

IF loadBalance < tolerance THEN

Determine workload of each object for this time step  
*(based on indicators or estimation from last time step)*

<table>
<thead>
<tr>
<th>Call partitioning method</th>
<th>4: Partitioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Migrate objects</td>
<td>3: Migration</td>
</tr>
</tbody>
</table>

END IF

Exchange ghost cells with neighbors  
2: Communication

Compute model equations  
1: Load balance

NEXT
Outline

- Introduction
- Dynamic Load Balancing
  - Objectives
  - Metrics: Workload, Load Balance
  - Typical Approach
- Partitioning Methods
- Software Stack
- Experiences with COSMO-SPECS+FD4
- Conclusion
Partitioning

- Partitioning = Assignment of objects to processes
  - Objectives of load balancing should be satisfied

Input:
- Number of processes $P$
- Weight of all objects $w_i$ (to optimize load balance)
- Information about neighborhood of objects (to optimize communication)
- Current partitioning (to optimize migration)

Output:
- Mapping of objects to processes
Partitioning: Classification of Methods

Partitioning Methods

Geometric Methods
- Need spatial coordinates and object weights

Graph-based
- Consider object decomposition as a weighted graph

Recursive Bisection

Space-Filling Curves

Global Graph-based

Local Graph-based

Partitioning: Recursive Bisection

- Cut the grid in two equal weighted parts
- Apply this algorithm recursively for each part until number of desired partitions is reached
  - Processor count $\neq 2^n$: cut in more than 2 parts or cut in unequal parts
- Very fast, but moderate scalability
- Requires fine granularity to reach good balance
- Moderate optimization of communication costs
- Versions:
  - Recursive Coordinate Bisection (RCB)
  - Unbalanced Recursive Bisection (URB)
  - Recursive Inertial Bisection (RIB)
Partitioning: Space-Filling Curves (SFCs)

- **Hilbert Curve**
  - 1D traversal of the grid
  - nD → 1D mapping / ordering

- **Morton Curve**
  - Data locality
    - Points close on the curve are also close in the nD grid
  - Self-similarity
    - Constructed recursively from a start template in $O(\log n)$

- Most prominent for load balancing:
  - Hilbert curve (higher locality)
  - Morton curve (faster)
Partitioning: Space-Filling Curves (SFCs)

- Partitioning is reduced to 1D
- 1D partitioning is core problem of SFC partitioning
  - Decompose object chain into consecutive parts
- Two classes of existing 1D partitioning algorithms:
  - Heuristics: fast, parallel, no optimal solution
  - Exact methods: slow, serial, but optimal
- SFC implicitly optimizes for low communication and migration
  - SFC locality leads to moderate communication costs
  - Migration typically between neighbor ranks


Partitioning: Space-Filling Curves for Mesh Refinement

- Space-Filling Curves are well suited for structured adaptive mesh refinement (AMR) due to their self-similarity

Start template | Refine | Refine | Partition

Partitioning: Global Graph-based Methods

- View the decomposition as a weighted graph
  - Vertex weight: object's workload
  - Edge weight: comm. costs between objects

- Works for irregular grids
- Very good optimization of communication costs
- Very time consuming, hard to parallelize efficiently
- High migration costs

- Different heuristics / many publications
  - Greedy graph partitioning (fast, but worse quality)
  - Recursive spectral bisection (very slow)
  - Multilevel graph partitioning (widely used)
Partitioning: Multilevel Graph Partitioning

Multilevel hypergraph partitioning

- Edges connect more than two nodes
- Accurate model of communication and migration costs leads to higher quality
- More expensive

Multilevel + coordinate mapping + geometric method (ScalaPart)

- Graph is mapped to a grid to get coordinates of vertexes
- Fast geometric method + border refinement
- Much better scalability


Partitioning: Local Graph-based Methods

- Only subsets (i.e. neighborships) of existing partitions exchange objects
- Requires an initial partitioning
- Requires multiple iterations (with different subsets) to reach good balance
- Sufficient for small workload changes or as refinement step for other methods
- Typically very fast, but depends on number of iterations
- Scalable by design: only local actions

Algorithms
- Diffusion algorithms
- Work-stealing algorithms
Partitioning: Hierarchical Methods

- Organize processes in hierarchy
  - I.e. derived from network or application topology
- Apply partitioning method independently in each level
- Better scalability than centralized approaches
- Less memory requirements than (serial) methods
- Most promising methods for large scale


Partitioning: GrapevineLB Distributed Load Balancer

- Does not fit in classification
  - Does not use communication topology information

- Local migration decisions based on knowledge about some underloaded processes
  - Information is spread with a randomized epidemic (gossip) algorithm, only a few rounds
  - Every overloaded process knows about some randomly chosen underloaded processes

- Objects are transferred to random processes that are known to be underloaded
  - They may reject the object if they already received enough load

- Runtime comparable to diffusion, but much better load balance

Partitioning: Scalability Challenges

- Large number of processes and objects
- Serial algorithms not sufficient
  - Large memory and network usage when collection weights of 1M-1G objects at one process
  - Even the simplest heuristic would be too slow
- The challenge is to find algorithms that
  - Leave weights distributed or communicate them only sparsely (e.g. within neighborship)
  - Nevertheless achieve global balance (without a detailed global view)
Outline

- Introduction
- Dynamic Load Balancing
  - Objectives
  - Metrics: Workload, Load Balance
  - Typical Approach
- Partitioning Methods
- **Software Stack**
- Experiences with COSMO-SPECS+FD4
- Conclusion
Dynamic load balancing in HPC applications is usually hand-coded in the application.

Huge coding effort when introducing load balancing to a big/real HPC application.

3rd party libraries to compute partitioning:
- ParMetis: multilevel graph, diffusion, multiconstraint
- Jostle, PT-Scotch, DibaP: multilevel graph
- Zoltan: geometric, hypergraph, hierarchical, can use ParMetis and PT-Scotch
Software Stack: Runtime / Framework Layer

- MPI is static, no load balancing
- MPI-based frameworks
  - Frameworks for parallel PDEs: PETSc, FD4, ...
  - Adaptive mesh refinement frameworks: ALPS, GrACE, Chombo, Racoon, ...
- Load balancing of virtual MPI processes: Adaptive MPI
- Alternative runtime systems: Charm++, PREMA

Huang et al., Performance Evaluation of Adaptive MPI, PPoPP 2006

Acun et al., Parallel Programming with Migratable Objects: Charm++ in Practice, SC 2014

Charm++ system view
https://charm.cs.illinois.edu/tutorial/CharmRuntimeSystem.htm
Software Stack: Operating System Layer

- Typical HPC system: OS reduced as much as possible
- Single-System Image (SSI) OS's allow load balancing and transparent process migration in a cluster
  - Used for load balancing between different applications, but not within an application
- Systems
  - Kerrighed, (open)Mosix, OpenSSI
- Few installations with ~100 nodes
- No experience with large state-of-the-art HPC systems
- FFKM seeks to migrate (oversubscribed) MPI processes for load balancing

Outline

- Introduction
- Dynamic Load Balancing
  - Objectives
  - Metrics: Workload, Load Balance
  - Typical Approach
- Partitioning Methods
- Software Stack
- **Experiences with COSMO-SPECS+FD4**
- Conclusion
COSMO-SPECS+FD4: Parallelization and Coupling Concept

COSMO Atmospheric Model

Model Coupling

Cloud Microphysics Model

Large (legacy) Codebase

2D Decomposition

Static Partitioning

Block-based 3D Decomposition

Dynamic Load Balancing
COSMO-SPECS+FD4: Space-filling Curve vs. Graph Part.

- **SFC achieves better load balance**
  - [Graph showing load balance comparison between SFC and ParMetis]

- **ParMetis reduces communication better**
  - [Graph showing communication volume comparison between SFC and ParMetis]

- **SFC migration is faster at large scale***
  - [Graph showing migration time comparison between SFC and ParMetis]

- **SFC computes much faster**
  - [Graph showing partitioning time comparison between SFC and ParMetis]

* due to local communication pattern that leads to less network usage & contention
COSMO-SPECS+FD4: SFC Partitioning with Heuristic
Large scale applications require a fully parallel method, i.e. without gathering all task weights

- Run parallel H2 to create G < P coarse partitions:
  - Orig part.
    - Parallel Heuristic H2
      - P0 / P1
      - P2 / P3
      - P4 / P5
      - P6 / P7
  - Coarse part.
    - P0 / P1
    - P2 / P3
    - P4 / P5
    - P6 / P7

- Run G independent instances of exact QBS* (q=1.0) to create final partitions within each group:
  - QBS*
    - QBS*
    - QBS*
    - QBS*
  - Final part.
    - P0
    - P1
    - P2
    - P3
    - P4
    - P5
    - P6
    - P7

Parameter G allows trade-off between scalability (high G → heuristic dominates) and load balance (small G → exact method dominates)

H2 nearly optimal if $W_{\text{max}} \ll W_N / P$: Miguet, Pierson, Heuristics for 1D rectilinear partitioning as a low cost and high quality answer to dynamic load balancing, LNCS, vol. 1225, 1997, pp. 550-564.
<table>
<thead>
<tr>
<th>Process</th>
<th>Time</th>
<th>Time</th>
<th>Time</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>313.860578 s</td>
<td>+0.25 s</td>
<td>+0.50 s</td>
<td>+0.75 s</td>
</tr>
<tr>
<td></td>
<td>27208</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27216</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27224</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27232</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27240</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27248</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27256</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27264</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27272</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27280</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27288</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27296</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27304</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27312</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27320</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27328</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27336</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27344</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27352</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27360</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27368</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27376</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27384</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27392</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27400</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27408</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27416</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27424</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27432</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27440</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27448</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27456</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27464</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27472</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27480</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27488</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27496</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27504</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27512</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27520</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27528</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**COSMO-SPECS+FD4: Comparison of Partitioning Time**

- **ExactBS**: exact method, but slow and serial
- **H2**: fast heuristic, but may result in poor load balance
- **HIER**: hierarchical algorithm implemented in FD4, achieves nearly optimal load balance

---

**Graph**

- **ExactBS**: 2668 ms
- **QBS**: 692 ms
- **H2seq**: 363 ms
- **H2par**: 40.5 ms
- **HIER\*\(_{G=64}\)**: 8.55 ms
- **HIER\*\(_{P/G=256}\)**: 3.77 ms

Balancing 1,357,824 objects, IBM Blue Gene/Q
COSMO-SPECS+FD4: Comparison of Load Balance

- ExactBS: exact method, but slow and serial
- H2: fast heuristic, but may result in poor load balance
- HIER*: hierarchical algorithm implemented in FD4, achieves nearly optimal load balance


Balancing 1 357 824 objects, IBM Blue Gene/Q
Outline

• Introduction
• Dynamic Load Balancing
  – Objectives
  – Metrics: Workload, Load Balance
  – Typical Approach
• Partitioning Methods
• Software Stack
• Experiences with COSMO-SPECS+FD4

Conclusion
Load balancing is important for many HPC applications
Will get more important in future
  – Models get more complicated → load variations
  – Hardware gets more complicated → capacity variations
Quest for high-quality and highly scalable dynamic load balancing methods
  – We will see more hierarchical and fully distributed methods
Application developers need better support
  – Use (domain-specific) frameworks?
  – Replace (much too static) MPI by new runtime?
  – Get help from OS?
Thank you very much for your attention!

Acknowledgments
Verena Grützun, Ralf Wolke, Oswald Knoth, Martin Simmel, René Widera, Matthias Jurenz, Matthias Müller, Wolfgang E. Nagel

Funding

Leibniz Supercomputing Centre of the Bavarian Academy of Sciences and Humanities

www.vampir.eu

PICon
GPU

www.cosmo-model.org

picongpu.hzdr.de

Deutsche Forschungsgemeinschaft

Europa fördert Sachsen.

IMData

Center for Information Services & High Performance Computing

ZIH