Load balancing bibliography

Alan Heirich and Karthik Murthy

May 18, 2018

1 Categories

1.1 introduction

- Unbalanced tree search benchmark used by lifeline mapper [42]. Is there a standard benchmark for AMR simulations?
- Does the LB algorithm depend on the application? Does it depend on the programming model (Legion)? Is it independent of both?

1.2 Category axes

- local vs. global: does the entire computer system have to participate, or can a small subset balance locally?
- nearest neighbor vs. long range exchanges
- static vs. dynamic
- expensive vs. cheap (related to static vs. dynamic)
- embarassingly parallel vs. interconnected: does mapping matter?
- continuous domain (mesh simulations) vs. discrete (tree search)

1.3 diffusion

[22]

notes on [22] from Alan
finite element, finite volume unstructured adapting meshes
Diffusive partition improvement, application specified criteria
N-graph: hyper graph structure represents relations among elements
Diffusion is performed on the N-graph
A multigraph allows multiple edges between a pair of vertices
The N-graph does this for hyper graphs
Saves on memory versus just a graph
Imbalance = T_max/T_mean
Does not explain how to compute the transfer amount
Graph distance: migrate elements in order according to their distance from the cell center
Experiments : billion element mesh airplane tail structure
Argonne Mira blue gene Q
$128 * 2^{10}$ to $512 * 2^{10}$ elements cases
Showed reasonable improvement, 1.5 imbalance to 1.12
Did worse on larger problems

[29]
challenges:
- rebalance dynamically
- balancing based on entities
- diffusion with a local-exchange, i.e., bottom-up is bad
pros:
-log n approach since its top down
-communication cost accounted for, in a way
cons:
-a binary tree approach
-assume work load characterization
NOTE: Add [5] reference to the collection, its asurvey paper on load balancing

notes on [29] from Alan
fea unstructured adaptive mesh
No topology assumptions
Multilevel algorithm complexity is logarithmic in number of processors
Diffusion methods may require many iterations, see Boillat claims of $O(n^2)$ iterations on n processors
Claim: pairwise diffusion can result in large load imbalance (I dont think this is true of Laplace iteration although low frequency disturbances subside slowly) See Cybenko claim of $log_2(n)$ steps iteration, uses hypercube topology
The algorithm here achieves $O(log_2 n)$ but does not depend on topology
Local communication costs less than nonlocal : hypercubes (this will always be true, just how much)
Load balancer should respect existing adjacency relationships of the domain
topology of the mesh may not match topology of computers
Basic diffusion method pairwise exchange of $0.5 * (l_i - l_j)$ units of work
Multilevel algorithm aims to eliminate large scale imbalances
At each level divide processors into two sets and balance them as with two individuals
No explanation of how to choose these sets
Proof that it takes $log_2(n)$ steps duh
requires entire system rebalance at once, not a local method
Claims standard diffusion techniques are bad because they dont guarantee number of iterations
[20]

notes on [20] from Alan
Efficient cell selection scheme
Local and global diffusion schemes, global performs best
Global means knows all servers, local means only knows nearest neighbors
Note: reference ou and ranka 1997 solve lb problem as linear programming
Distributed virtual environments prefer fast solution over optimal solution
Experiment using simulated workloads, virtual environment users moving through the environment, environment is partitioned into regions, one region per server
[30]
challenges:
-diffusion vs geometrical vs graph-based methods
-tasks are migrated based on geometrical coordinates or graph topolgogy
-geometrical good balance but lot of migration, e.g., gossip
-diffusion never really considered in HPC
this is a survey paper[a very good reference]:
- load transfer vector prepared by node to move tasks to another node
- CHEBY algorithm not uses difference in work load
- PARMETIS uses diffusion, ZOLTAN library and GLB library should be in mind
- chemotaxis uses capacity on me - load on target
- which tasks to send based on comm reduction, i.e., comm with neighbors only

their questions are our questions:
-how can we apply diffusion on todays common hardware topologies like fat trees?
-What is a fast and high-quality method for task selection allowing to trade-off balance, migration and edge-cut?
-How do we scalably implement the termination criterion for diffusion when no fast collectives are available? E. g. use approx or auto-tune iteration count?

notes on [30] from Alan
Good survey paper, worth reading again
Compare diffusion to geometric and graph based methods on thousands of nodes
Space filling curves, recursive bisection, parMetis, hierarchical space filling curves
Concludes diffusion has advantages
. The second-order algo- rithm [15] extends OD such that the previous iterations transfer influences the current. The parameter (0,2) controls the influence. Optimal values are derived in [9]
. [9] R. Elsa sser, B. Monien, and R. Preis. Diffusion Schemes for Load Balancing on Heterogeneous Networks. Theory Comput. Sys., 35(3):305320, 2002.
. [15] S. Muthukrishnan, B. Ghosh, and M. H. Schultz. First and Second Order Diffusive Methods for Rapid, Coarse, Distributed Load Balancing. Theory Comput. Sys., 31:331354, 1998.
Survey based on improving diffusion
Original diffusion
Second order diffusion
Improved diffusion called cheby
Chemotaxis-inspired diffusion, additional round of exchange of capacity of the target node guides the diffusion locally
Dimension exchange - Xu and Lau
[41]

notes on [41] from Alan
Our contribution can be summarized as follows: we give a direct explicit expression of the balancing flow generated by a generalized diffusion algorithm and we show that this flow has an interesting property, that it is a scaled projection of any other balancing flow in the same heterogeneous environment. We give estimations for the second largest eigenvalue of a generalized diffusion matrix and we estimate the complexity of the proposed algorithm. We further show that this algorithm has a better convergence factor than the hydrodynamic algorithm [17,18]. Compared to other approaches, the one we consider here offers the advantage of not using parameters that are dependent upon the eigenvalues of the Laplacian of the communication graph.
Solves load balancing and mapping
Since communication changes so frequently cannot afford to compute Laplacian eigenvalues
Analogy to markov chains
Connections between generalized diffusion matrices and Laplacian spectrum of the graph
Bounds on eigenvalues
Migration flow - expression for the flow
Good paper lots of analysis
Estimations were given for the maximum number of steps that such an iterative process may take to balance purpose, some general bounds were formulated for the second largest eigenvalue of a generalized diffusion matrix. These bounds were also used to show that there are generalized diffusion algorithms that theoretically converge faster than the hydrodynamic algorithm
[28]
[18]
[49]

notes on [49] from Alan
uses successive over relaxation to find an optimal step size for convergence
I wonder: do these convergence issues really matter? Is the simplest scheme good enough?

[12]

notes on [12] from Alan

Show polynomial time convergence to equilibrium

Remark 5. In the discrete case, i.e. working with individual processes, our problem is equivalent to the random walk problem in graphs[20] 20. D. Aldous. An inaoduction to covering problems for random walks on graphs.J. Theoretical Probability, 2(1), 87-89 (1989).

[50] challenges:
-mapping + load balancing at the same time
-a form of diffusion is the answer for both
-goals of mapping, workload balance and comm time
pros:
-based on the laplacian matrix of communicating processes
-delay and fault tolerant, insensitive to problem scale, convergence depends on initial conditions (hmm)
cons:
-np-complete problem
-m to be 2, planar connection networks, but arbitrary topologies like hierarchical, maybe linearization, not sure: did not understand the mapping to torus by a broadcast-type strategy, what about binomial dissemination?.
-reduction of the problem for a specific instance ? isnt it ?

[44]

1.4 game theory

[26]

notes on [26] from Alan

Static load balancing problem

Noncooperative game: processors work independently to arrive at equilibrium

Characterize Nash equilibrium and derive greedy algorithm to compute it

Assume Poisson arrivals, exponentially distributed task times

ϕ_i job generating rate at node I

s_{ji} fraction of user j tasks to be sent to node i

μ_i processing rate at node i

Load balancing strategy for user j is a vector of s_{ji}

Minimize response time for user j

Remark: this is for multiple users (humans) submitting jobs to a distributed system (cluster).

Our case is one user (application) submitting tasks to an exascale system

Best reply for a user is a strategy that gives minimal response time for that user in light of other users strategies

Similar problem for one user treated in Tang and Chanson[35] X. Tang, S.T. Chanson, Optimizing static job scheduling in a network of heterogeneous computers, in: Proceedings of the International Conference on Parallel Processing, August 2000, 373382. Not very interesting.

Equation (8) defines the BEST_REPLY solution

Execution time is O(n log n) due to the need to sort computers by workload

Otherwise it would be O(n)

This algorithm requires global knowledge of the workload of every computer at every agent and knowledge of all users strategies

Not scalable

Compared to two other lb schemes, one does a global optimization based on global knowledge, and an IOS scheme that gives good quality results

Experiments on 16 node cluster

[1]

notes on [1] from Alan

Establish uniqueness of Nash equilibria

No software experiment

Not relevant

[7]

notes on [7] from Alan

Entirely theoretical result, random movement of tasks with uniform weights

[6]

notes on [6] from alan

Load Rebalancing Games in Dynamic Systems with Migration Costs Initial assignment of jobs to identical parallel machines

Machines are added or subtracted

Extension parameter σ is added to the cost of a job after it is moved (this is intended for a one time cost, not repeated)

Paper proves existence and calculation of Nash equilibrium and Strong Nash equilibrium

Under some assumptions any stable modified schedule approximates well an optimal schedule

Each job incurs a cost which is equal to the total load on the machine it is assigned to ????

See game theoretic treatments of this problem 12, 2, 5, 8, survey in 17 . [2] N. Andelman, M. Feldman, and Y. Mansour. Strong Price of Anarchy. In SODA, 2007

. [5] A. Czumaj and B. V ocking. Tight bounds for worst-case equilibria. In ACM Transactions on Algorithms, vol.3(1), 2007

. [8] A. Fiat, H. Kaplan, M. Levi, and S. Olonetsky. Strong Price of Anarchy for Machine Load Balancing. In ICALP, 2007.

. [12] R.L. Graham. Bounds on Multiprocessing Timing Anomalies. SIAM J. Appl. Math., 17:263269, 1969.

. [17] B. V ocking. In N. Nisan, T. Roughgarden, E. Tardos and V. Vazirani, eds., Algorithmic Game Theory. Chapter 20: Selfish Load Balancing. Cambridge University Press, 2007.

s_0 assignment of n jobs on m_0 machines m machine are added or removed

Seek a Nash equilibrium s where no machine can improve its situation by changing machines

"Assume that some preprocessing is done at the time a client is assigned to a server, before the download actually begins (e.g., locating the required file, format conversion, etc.). Clients might choose to switch to a mirror server. Such a change would require repeating the preprocessing work on the new server. Another example of a system in which extension penalty occurs is of an RPC (Remote Proce- dure Call) service. In this service, a cloud of servers enables service to simultaneous users. When the system is upgraded, more virtual servers are added. Users might switch to the new servers and get a better service (with less congestion), however, some set-up time and configuration tuning is required for each new user. Note that in all the above applications, the delay caused due to a migration is independent of the migrating job.

These sound like one-time costs to me hence my objection to this analysis, also the cost may depend on the job (amount of data movement, size of code, number of open files, etc)

Section 1.1 some issues with notation, use of $L_i(s)$ and $L_s(j)$ to mean different things? Price of anarchy = ratio between max cost of a Nash equilibrium and the optimum schedule (load imbalance) Price of stability = ratio between min cost of a Nash equilibrium and the optimum schedule

A set of players form a coalition if each job moves and strictly reduces its cost Assignment is a strong Nash equilibrium if there exists no such coalition Similar notions of strong price of anarchy, strong price of stability

"The simple greedy List-scheduling (LS) algorithm [11] provides a $(2 - 1/m)$approximation to the minimum makespan problem. A bit better approximation ratio of $((4/3) - (1/3m))$ is guaranteed by the Longest Processing Time (LPT) algorithm [12]. A PTAS for the minimum makespan problem on identical machines is given in [13].

"We show that any job scheduling game with added or removed machines possesses at least one Nash equilibrium schedule. Moreover, some optimal solution is also a Nash equilibrium, and thus, the price of stability is 1. We show that in general, the price of anarchy is unbounded when machines are either added or removed.

"We note that in a dynamic setting in which machines are added or removed and migrations are free of cost (i.e., when $\sigma = 0$), then the results known for classic load balancing games apply. In particular, the P oA assuming $\sigma = 0$ is $2 - (2/(m + 1))$ for a game with m machines in the modified $m + 1$ systems. The

proofs are identical to the proofs for a fixed number of machines. Thus, the difference between our results and the results for the classic load balancing game are due to the migration penalty.

Sections on machine addition, machine removed, coalitions. Each case proves existence of solutions.

[4]

Reoptimization of the minimum total flow-time scheduling problem

Guy Baram, Tami Tamir

Dynamic load balancing, accounts for migration cost Optimal algorithm to find optimal solution with minimal migration cost (transition cost) Minimal solution can be found by greedy algorithms, Shorted Processing Time(SPT) assign jobs in nondecreasing order by length Example applications: manufacturing systems with jobs migrated along production lines; live migration of running VMs in a cloud data center; RPC servers

Other papers that minimize the migration cost:

. [10] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, Live migration of virtual machines, in: The 2nd Symposium on Networked Systems Design and Implementation (NSDI), 2005.

. [19] S. Hacking, B. Hudzia, Improving the live migration process of large enterprise applications, in: The 3rd International Workshop on Virtualization Technolo- gies in Distributed Computing (VTDC), 2009.

J0 set of n0 jobs M0 set of m0 identical machines Pj is processing time for job j S0 schedule of the initial instance Changes include adding/deleting jobs from J0, machines from M0, and changing values of pj

A machine can process at most one job at a time Price list theta(i,i,j) is cost to migrate job j from machine I to machine I Job extension penalty delta(i,i,j) is time extension increase in pj after moving job j from machine I to I Cj completion time of job j Minimize sum Cj = total flow time

Two questions: 1. Reschedule to find minimum possible transition cost 2. Reschedule within a given budget B

Section 2, time t=0: optimal algorithm for rescheduling, complexity based on complete matching in bipartite graph with O(nm) vertices Section 3, time t¿0: Section 4: unit migration costs and no job extension penalty. Section 5: rebalancing with a budget Section 6: results and discussion

Section 2

. [7] J.L. Bruno, E.G. Coffman, R. Sethi, Scheduling independent tasks to reduce mean finishing time, Commun. ACM 17 (1974) 382387.

. [18] W. Horn, Minimizing average flow-time with parallel machines, Oper. Res. 21 (1973) 846847.

Problem instance: nxm matrix of job times, pij is time for job j on machine I Seems like a bug in the paper with indexing here, I is jobs and j is machines

Construct a bipartite graph, nodes V = J u U J is the set of n jobs U is the set of mn nodes

Node qik is the kth from last position on machine I Edges completely connect nodes in J to nodes in U Edge weights are w(vj, qik) = k pij If job j is the kth from the last job to run on machine I, it contributes k times pij to the sum of completion times

?? Why k times?

Optimal solution is found by min-weight matching in the bipartite graph. (Obvious)

*** this algorithm is not distributed, requires global knowledge and centralized computation, ergo not scalable

1.5 mapping

[39]

Heuristic algorithm: compute Fiedler vector, use it as an edge separator, then find the vertex separator by maximum matching in a subgraph

Require the two parts of the graph to be roughly equal in order to achieve load balance (not exactly equal)

Number of zero eigenvalue of laplacian matrix is equal to the number of connected components of the graph

Lots of definitions of Fiedler vector, laplacian matrix, spectral theory

4. Partition of grid graphs

Start with a path graph ?

Laplacian matrix is tridiagonal -¿ path graph is x 0 x diagonal pattern

5 spectral partitioning algorithm

Compute Fiedler vector

Use median value of vector elements to partition vector into left, right halves This is an edge separator

We require a vertex separator Simplest method is to choose the smaller of the two endpoint sets (vertices divided by the edge separator)

But we want to compute the smallest separator Solution is a vertex cover of the bipartite graph Remove vertices a,b such that every edge is incident on one or both of these vertices

To find a minimum vertex cover use maximum matching algorithm

Worst case time complexity for algorithm to find one partitioning is O(sort ne) Where e is number of edges, n is the length of the Fiedler vector

[15]

Hyper graph partitioning for dynamic lb. minimizes sum of communication costs plus migration cost.

Parallel multilevel repartitioning with Zoltan load-balancing toolkit

Results quality compressed favorably to ParMETIS

$t_total = alpha(t_compute + t_communication) + t_migration + t_repartitioning$

They ignore $t_compute$ and $t_repartuition$: for $t_compute$ they assume a balanced workload; for $t_repartition$ they assume this time is negligible

Net = hyper edge (edge with more than two vertices)

Vertices have weights wi

Nets have costs cj (like edge weights)

Partition the hyper graph - each partition corresponds to one computer

Hypergrah H, Partition P

Wp = sum of vertex weights in partition Vp

P=Vi. Partition consists of a set of parts

Connectivity λ_j of net j is the number of parts it connects

Multilevel partitioning - fine to coarse strategy, partition the coarse graph project back up

Their model combines communication cost with migration cost

Epoch = execution in time between two dynamic rebalance operations

Repartitioning hyper graph - augment the current epoch hyper graph Hj with additional vertices and nets to model data migration costs. Now partition the resulting hyper graph using fixed vertices.

$H^j = (V^j, E^j)$ is the hyper graph that models epoch j of the computation

Rebalance following each epoch

Construction of the repartitioning hyper graph: Add k new partition vertices ui, with zero weight, for each of k partition For each vertex v in the old hyper graph, add a migration net from v to new vertex ui if v was in partition I

Figure 1 is confusing read the text carefully

Vertex ui must be fixed to partition I

Now partition the new hypergraph

If the partition cuts a migration net then the cost of migration gets counted in the overall communication cost This is the key idea that accounts for migration cost

Parallel multilevel hypergdaph partitioning with fixed vertices

Hg partition is NP hard, approximate using multilevel heuristics

Like multigrid, coarsen the graph and partition the coarsest level

Propagate the partitions upward to the full graph

[27]

[43]

Concerned with scheduling when memory is limited.

For many parallel applications, the memory requirements can be significantly larger than for their sequential counterparts and, more importantly, their memory utilization depends critically on the schedule used when run- ning them.

Using the inspector/executor model, BMS tailors the set of allowable schedules to either guar- antee that the program can be executed within the given memory bound, or throw an error during the inspector phase without running the computation if no feasible schedule can be found.

Since solving BMS is NP-hard, we propose an approach in which we first use our heuristic algorithm, and if it fails we fall back on a more expensive optimal approach which is sped up by the best-effort result of the heuristic. Unfortunately, parallel execution is known to increase mem- ory requirements compared to a serial baseline [8,11]. .

[8] Robert D. Blumofe and Charles E. Leiserson. Scheduling multithreaded computations by work stealing. In: J. ACM (1999).

[11] F. Warren Burton. Guaranteeing Good Memory Bounds for Parallel Programs. In: IEEE Trans. Softw. Eng. (1996)

[16] .I. Dooley et al. A study of memory-aware scheduling in message driven parallel programs. In: HiPC. 2010

This problem is a general case of the register sufficiency problem (NP hardness comes from this)

Inspector/executor: inspector builds a task graph that shows parent-child task relationships and reader-writer relationship for the data (similar to Legion). Inspector identifies scheduling restrictions that lead to bounded-memory execution. Enforce these restrictions in the executor stage when the app runs a load balancing work-stealing scheduler.

Contributions:

Heuristic algorithms for bounded memory scheduling (BMS) based on inspector/executor Optimal algorithm based on ILP

Schedule memoization for heuristic algorithm

Concurrent Collections Programming Model (CnC)

Tasks (called steps)

Step tag - identify specific instance of a step (task)

items - dynamic single assignment variables

Item collections - identified by key, collect related items together like a struct

Step reads items by calling item_collection.get(key), blocks waiting for data They claim that building the task graph dynamically is too late for bounding memory consumption (?)

Many analyses of task-parallel programs (such as data race detection) require understanding the task-parallel struc- ture of the computation, which is usually unknown at com- pile time. As a result, many of these analyses build the task graph dynamically, while the application is running. Unfor- tunately, this is too late for certain optimizations, such as bounding the memory consumption of the program.

Expansion functions augment the CnC task graph to provide the necessary information

******* If the keys of items read and written and tags of steps spawned are only a function of the current step tag, then the application has independent control and data, which is needed to accurately model an application using BMS. ***** Sounds terrible!

If the keys and tags depend on the values of items, we say that the application has coupled control and data. When faced with an application with coupled control and data, one possible solution is to include more of the com- putation itself in the graph expansion functions. In the extreme case, by including all the computation in the ex- pansion functions, we would be able to obtain an accurate dynamic task graph. Unfortunately, in the worst case, the computation would be performed twice, once for the expan- sion and once for the actual execution. However, our expe- rience is that many application contain independent control and data, thereby supporting the BMS approach.

Assume all tasks have unit size, relax this in section 8

BMS heuristic algorithm is based on list scheduling

If no acceptable solution is found, run the optimal algorithm

5.1 successive relaxation of schedules Sort depth-first rather than breadth-first in the task graph (duh)

5.2 color assignment Determines concurrent chains of tasks

6. Optimal algorithm through ILP

8.1 supporting multiple item sizes Assigns memory locations, deal with fragmentation that results from different sizes

1.6 heterogeneous

[23]

Overcoming Load Imbalance for Irregular Sparse Matrices Flegar and anzt. IA^3 17 : Irregular Applications Architecture and Algorithms

GPU implementation of sparse matrix-vector product Uses COO format, row-column index of each element ELL format - store only nonzero, pad with zeros to give same vector length, for SIMD

This is not a load balancing or mapping algorithm. This is a matrix multiply algorithm that tries to be load balanced.

[52]

Dynamic Load Balancing on Multi-GPUs System for Big Data Processing

This paper presents a novel dynamic load balancing model for heterogeneous multi-GPU systems based on the fuzzy neural network (FNN) framework. The devised model has been implemented and demonstrated in a case study for improving the computational performance of a two dimensional (2D) discrete wavelet transform (DWT).

Chen et al. [7] proposed a task-based dynamic load balancing solution for multi-GPU systems that can achieve a near-linear speedup with the increase number of GPU nodes. Acosta et al. [8] had developed a dynamic load balancing functional library that aims to balancing the load on each node according to the corresponding system runtime. However, these pilot studies are based on the assumptions that all GPU nodes equipped in a multi- GPU platform have equal computational capacity. In addition, the task-based load balancing schedulers these approaches relied upon fall short to support applications with huge data throughputs but limited processing function(s) since there are very few tasks to schedule, e.g. DWT.

. [7] L.Chen,O.Villa,S.Krishnamoorthy,andG.R. Gao, Dynamic Load Balancing on Single- and Multi-GPU Systems, Ipdps, 2010.

. [8] A.Acosta,R.Corujo,V.Blanco,F.Almeida,H.P. C. Group, and E. T. S. De Ingenier, Dynamic Load Balancing on Heterogeneous Multicore / MultiGPU Systems, 2010.

Their algorithm: divide work into equal size units; dynamically schedule those units according to evolution of dynamic workload; use fuzzy neural network to do this (?)

Fuzzy neural network predicts execution time for each GPU based on: flops rate; memory size; parallel scaling; occupancy rate of compute resources; occupancy rate of global memory. Train neural network based on historical records. (Main advantage seems to be that it allows GPUs to have different capacities).

CASE STUDY - discrete wavelet transform Same amount of work per work-item. Study used two different GPUs of different capacities. Only experimented with two GPUs.

I do not think this paper is a good idea.

[13]

A performance, power, and energy efficiency analysis of load balancing techniques for GPUs

Regular and irregular workloads, hard to give good performance in both cases. Dynamic multi-phase workload partition and work item-to-thread allocation. Low complexity, good results. Paper compares this to several other state of the art GPU lb algorithms.

NV Maxwell GTX 980, NV Jetson Kepler TK1 (low power embedded). Work-units are grouped into work-items. BFS graphs: work-units are graph vertices; work-items are neighbors of each work-item. Prefix-sum-array holds offset to each work-item.

PRIOR ART:

Static mapping:

[4] P. Harish and P. J. Narayanan, Accelerating large graph algorithms on the GPU using CUDA, in Proceedings of the 14th International Conference on High Performance Computing, ser. HiPC07, 2007, pp. 197208. Simplest, one work item per thread, not load balanced

[5] S.Hong,S.K.Kim,T.Oguntebi,andK.Olukotun,AcceleratingCUDA graph algorithms at maximum warp, in Proceedings of the 16th ACM Symposium on Principles and Practice of Parallel Programming, ser. PPoPP 11, 2011, pp. 267276. Virtual warp (group of threads): workload assigned to threads of same group almost equal, therefore reduce branch divergence and improve memory coalescence. Have to size the virtual warp correctly to get performance, this is a static parameter. Good lb within a warp but not balanced across warps.

Semi-dynamic mapping:

[6] F. Busato and N. Bombieri, BFS-4K: an efficient implementation of BFS for kepler GPU architectures, IEEE Transactions on Parallel Distributed Systems, vol. 26, no. 7, pp. 18261838, 2015. Dynamic virtual warps: VW calculated at runtime, lb within a warp and across warps.

[7] D. Merrill, M. Garland, and A. Grimshaw, Scalable GPU graph traversal, in Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ser. PPoPP 12, 2012, pp. 117128. Perfect balance among threads and warps, but expensive strip-mine first step, not worth it for regular workloads, good for irregular.

Dynamic mapping:

All require binary search across prefix-sum array

[8] A. Davidson, S. Baxter, M. Garland, and J. D. Owens, Work-efficient parallel gpu methods for single-source shortest paths, in Parallel and Distributed Processing Symposium, 2014 IEEE 28th International. IEEE, 2014, pp. 349359. Load prefix-sum chunks to GPU shared memory, process on GPU, do not guarantee balance across blocks, nor memory coalescing.

[9] O. Green, R. McColl, and D. A. Bader, Gpu merge path: a gpu merging algorithm, in Proceedings of the 26th ACM international conference on Supercomputing. ACM, 2012, pp. 331340. [10] Modern gpu library. [Online]. Available: http://nvlabs.github.io/ moderngpu/ 9 and 10 propose similar methods, two phases: partition and expand. First partition prefix-sum array into balanced chunks. Second expand all threads load their chunks into shared memory, each thread binary searches the prefix-sum array to get the first assigned work unit. Balanced across threads, warps and blocks at the cost of two binary searches. Problem: memory accesses of the threads to work-units is badly scattered in memory.

[12] F. Busato and N. Bombieri, A dynamic approach for workload partitioning on GPU archi- tectures, IEEE Trans. on Parallel Distributed Systems, vol. preprint, no. 99, pp. 115, 2016. Multiphase: like two phase but at lower cost.

A. Multi-Phase Technique

Hybrid partitioning phase: each thread searches work-items switching between optimized binary search and interpolation search. Iterative coalesced expansion phase: all threads load chunks into shared memory; each thread does optimized binary search to get the work-unit; three iterative sub-phases reorganize memory access for better coalescence.

1 writing on registers

2 shared memory flush, data reorganization

3 coalesced memory access

Steps 2 and 3 repeat until all are processed. Results perform better due to improved memory coalescence.

Results (lots of data): Their method how lowest execution time and lowest energy use out of 12 methods. Next best was virtual warps.

[16]

2008 graphics hardware On dynamic load balancing on graphics processors

Task weight not known in advance, new tasks created dynamically during execution. Compare 4 dlb methods, 1 is lock based. Test problem is octree partition of particles. Result: synchronization is very expensive, lock-free is better.

4 dlb methods: Centralized blocking task queue (lock based), Centralized non blocking task queue, Centralized static task list, Task stealing.

Thread level scheduling. Gpu thread block: equal size, all on one processor, fast local memory, barrier. Warp: 32 consecutive threads. Very old processors: 9600GT 512MB 64 cores.

Task stealing performed best.

[46]

A Load Balancing Strategy for Monte Carlo Method in PageRank Problem. PageRank can be solved by Monte Carlo (known result). Implement PageRank on GPU, deal with instruction divergence. Adopt the low-discrepancy sequences to simulate the random walks in PageRank computations Each thread of a block to compute a random walk of each vertex with a same low-discrepancy sequence

PageRank: list the relevant pages in order. Its a stationary vector of a random walk that simulates the process of surfing the internet. Interpret as frequency that a random surfer browses the web page, similar to popularity.

Connectivity matrix P defines all hyperlinks. P-ij = 1/k if page has k outgoing links and j is one of the links, = 1/n if page has no outgoing links, Else = 0. Surfer will choose next page from one of the links

at random with p=c otherwise randomly from the web with p=(1-c). So this is a Markov process with transition matrix

PP = c P + (1 - c) (1/n) E

Damping factor c = 0.85, E is matrix of all ones. The PageRank is the stationary distribution of the Markov chain, row vector pi such that pi PP = pi , pi 1 = 1

Linear algebra methods compute pi using power method iteration, pi := pi PP, approximately linear in n. Monte Carlo methods are faster and highly parallel. This paper considers efficient Monte Carlo on GPU.

Optimization issues:

Instruction divergence

Load balancing between threads of a warp

Memory access conflicts between threads

Cache performance among threads

Local memory utilization

Instruction divergence in Monte Carlo occurs because random numbers are different so Markov chains diverge. So they replace random sequences with low discrepancy sequences that are repeatable and identical, they reduce the variance of Monte Carlo Sampling [9-11]

9. Cervellera, C., Macci, D.: Low-discrepancy points for deterministic assignment of hidden weights in extreme learning machines. IEEE Trans. Neural Netw. Learn. Syst. 27(4), 891 896 (2016)

10. Gan, G., Valdez, E.A.: An empirical comparison of some experimental designs for the valuation of large variable annuity portfolios. Dependence Model. 4(1) (2016)

11. Zapotecas-Martnez, S., Aguirre, H.E., Tanaka, K., et al.: On the low-discrepancy sequences and their use in MOEA/D for high-dimensional objective spaces. In: 2015 IEEE Congress on Evolutionary Computation (CEC), pp. 28352842. IEEE (2015) We propose a divergence avoidable strategy to allocate the threads of a block to compute a random walk of each vertex with identical low-discrepancy sequence. Hence, the threads of a block will simultaneously execute same instruction to compute next state of Markov chain or terminate. We can address the instruction divergence by our strategy.

The static load balancing strategy pre-computes the length of each Markov chain to reduce the cost of absorbing state determination. The dynamic load balancing strategy dynamically determines the length of Markov chains to avoid the cost of loop condition determination. We will use experiments to test the efficiency of two strategies.

Another optimization is that we load the low-discrepancy sequences into shared memory to speed up data fetch operations.

The experiments indicate the bottleneck of our strategy is the memory access conflicts between threads. we store the adjacency matrix of a graph as the Compressed Sparse Row (CSR) format Since each random walk can be simulated independently, we can easily assign the computation of each walk for each thread. Furthermore, the all random walks begin from same node will be assigned to one block and all threads in a block simulate all random walks starting from one vertex in parallel. the time performance of a block will be critically influenced by the thread with longest random walks. So we try to use the determinacy of quasi random numbers to solve this problem.

Static load balancing

Wk, the expectation number of random walks whose length is k.we can call T kernels which have constant number of threads, Wk, each kernel computes k steps so that random walks in block have same length, in other words under same workload. all threads in a block have same workload, as a result, the warp execution efficiency will be improved. Their experiments show the dynamic lb made a minor improvement to the static lb case.

This is a good paper.

[40]

Dynamic Load Balancing Strategies for Graph Applications on GPUs

Existing work: node-based work-assignment to threads, gives poor load balance; edge-based requires lots of memory. New work: three improvements to ameliorate these problems

Breadth-first search BFS; Single source shortest past SSSP; Minimum spanning tree MST; Betweenness centrality; Graph500 benchmark database.

Node-based assignment leads to load imbalance if the degree of the nodes is not uniform, works well for data in Compressed Sparse-Row (CSR) format.

Edge-based dont have this problem but require reformatting the data to Coordinate List (COO). Takes too much storage to handle large graphs on limited GPU memory. Edge based requires distributivity in the operator which is not always possible,

Three contributions: Workload decomposition - assign edges, but only for nodes on the active list Node splitting - split a high degree node into multiple low degree nodes Hierarchical processing - hierarchy of work lists

Evaluated on BFS and SSSP algorithms

Workload decomposition:

In this approach, the processing elements in the worklist continue to be the nodes, but the workload of the nodes, namely, the edges, are decomposed across threads using a block distribution. E number of graph edges are partitioned across T threads such that each thread receives a contiguous chunk of E/T edges for processing. Thus, a given thread processes a subset of edges corresponding to a subset of nodes and all the edges outgoing from a node may not be processed by the same thread.

An advantage of workload decomposition is that it works with the CSR format and therefore, has a lower space com- plexity. A drawback of the workload decomposition is that it can lead to uncoalesced accesses since a nodes edges may get separated. In our experiments, we observe that the limitations of work- load decomposition affect its performance for large-diameter graphs (such as the road networks) but the method performs very well for scale-free graphs such as the social network

Node splitting

node splitting preprocesses the graph to split each high-degree node into multiple low-degree child-nodes. node-splitting approach has the advantage that it can work with the space-efficient CSR representation. A salient feature of our node splitting strategy is to automatically determine the threshold MDT for node splitting. Obvious methods based on a threshold or max-degree etc. do not work in general. we use a histogram based method in which we use HistogramBinCount number of bins representing the ranges of out-degrees of the nodes in the original graph.

An advantage of the node splitting approach is that it con- tinues to work with the space-efficient CSR format. all the edges of a node are processed by the same thread, reducing bookkeeping and improving the scope for memory coalescing. The primary disadvantage of node splitting is that it results in extra atomic operations to update the child nodes whenever the parent node gets updated. A secondary disadvantage is the overhead of computing the histogram to find the MDT.

node- splitting provides considerably better load-balancing. In ad- dition, it provides comparable performance for large diameter graphs (such as road networks); but it has a high overhead for power-law degree distribution graphs.

Hierarchical processing

Hiearchical processing performs a time-decomposition of the workload. It achieves this by partitioning the main (super) worklist into several sub-worklists. If the sub-worklist is large, it can be further partitioned into sub-sub-worklists, and so on. This builds a hierarchy of worklists. The depth of this hierar- chy is tunable, and we utilize the histogram-based approach in node-splitting (Section III-B) for finding the maximum degree threshold (MDT) which determines when to split a worklist into sub-worklists.

Compared BFS and SSSP to implementing from LonestarGPU which uses node-based distribution.

1.7 large scale

[34] [8] [53] [21] [37] [5]

1.8 task based

[2]
[9]
Locality-aware task scheduling for homogeneous parallel computing systems - LeTS heuritic (locality of data + load balancing)
- working task group formation phase
- capture reuse of data across tasks
- working task group ordering phase

- minimzes reuse distance of shared data between tasks

-NOTE: offline list scheduler takes the app profile (task graph), generates schedule based on that.
-IMPORTANT: Locality-aware task management for unstructured parallelism: a quantitative limit study in SPAA 2013.
—- this first introduced this notion
—- 2x over randomized work stealing
—- cache levels + cache private/shared control factor in local scheduling.
—-METIS divides the vertices from a task sharing graph into a given number of groups, while trying to (a) maximize the sum of edge weights internal to each group (i.e., data sharing captured by a task group), and (b) equalize the sum of vertex weights in each group (i.e., balance load).
—-a task group should be formed so that the working set of the group fits in cache.
—-task groups should be formed so that the cache size falls between the union and sum of task footprint
—-given a shared cache line, the sharing degree denotes the average number of tasks within a task group that share it.
—-sharing degree indicates the potential impact of task ordering. Specifically, a high sharing degree implies that data is shared by a large fraction of tasks.
—-clustered sharing: The sudden increase in cut cost means that the task group size became small enough that tasks sharing their key data structures have been separated into different groups. Ordering those task groups so that they execute consecutively will increase locality.
—- showcase improvements by varying CCR (compute-comm ration), task graphs and num of cores.
[54]

[38]
Hybrid work stealing of locality-flexible and cancelable tasks for the APGAS library
- based on lifelines + forkjoin pool (java)
- cancellable asyncAny tasks; hmm.
- identifies an opportunity of corrected load balancing, is it the same as a task migrating twice
- hybrid schemes in general
- able to obtain near linear speedup

Legion does not have this
- cancellation is useful in search tasks.
- nature of cancellable tasks are that they are independent

phases of balance+work
work stealing + proactive + reactive

[3]
Handling Transient and Persistent Imbalance Together in Distributed and Shared Memory - transient
- persistent
- heterogenous clusters intra- and inter-node
- chare objects then adaptively share work with other cores in the same process, exposing fine-grained tasks only to the extent that otherwise idle cores are available to help execute them.
- [32]
[19]
[24]

1.9 Work Stealing

[51] [17] [10] [11] [42]

1.10 other

[25] [14] [35] [48] [33] [31] [45] [36]

1.11 reviews

[47]

2 MEETING NOTES

2.1 what we are trying to do is expand on alan's equation

- roofline analysis
- work stealing
- expand mapping hierarchical
- work-first vs share-first policy, can this even be captured in a formula ?
- adding penalty for a failed steal
- accounting for critical path length
- with workload characterization
- effect of diffusive policies, update lv based on neighbor difference, second-order diffusion, improved diffusion,
- chemotaxis-inspired, dimension

- what about task selection (i.e., mapping) is it already covered

- graph partitioning, zoltan, parmetis

2.2 Some Thoughts

- in total, are we extending Diffusive methods to capture workloads without this laplacian angle ?, i.e., where the workload characterization does not follow a laplacian..

- in which case, do we need a result such as below ?
if requests are made randomly by P processors to P deques with each processor allowed at most one outstanding request, then the total amount of time that the processors spend waiting for their requests to be satisfied is likely to be proportional to the total number M of requests, no matter which processors make the re- quests and no matter how the requests are distributed over time
We can view each ball and each bin as being owned by a distinct processor. If a ball is in the reservoir, it means that the ball?s owner is not making a steal request. If a ball is in a bin, it means that the ball?s owner has made a steal request to the deque of the bin?s owner, but that the request has not yet been satisfied. When a ball is removed from a bin and returned to the reservoir, it means that the request has been serviced. Title of the paper: balls, bins and more

2.3 misc

- the task comm matrix (i.e., a dependence matrix of tasks) can be formed for regular and irregular, hence these techniques apply to both. -

2.4 comprehensive model.

- transfer of the tasks cost,
-in the diffusive models this transfer cost was not considered, but it could be a constant per task
-finding the task to transfer to.

- include the cost of pushing things to GPU, also AOS to SOA.
- critical path, control flow dependency graph
- who should run before whom is not there in the cost model ?
- graph that we have not accounted for, i.e., depenency graph.
- d, i.e., distance matrix between processors could change
- c, i.e., comm matrix between tasks could change, i.e., dimensionality could change

Nice to have - mapping dependencies, or is it?
- is not accounted for,
- penalty, if e.g., 2 to gpu1, 2 to gpu2.
- change in d and c is it equivalent to evaluating to the cost, i.e., not the same from the last step.
- theoritical proof of the eigen value is good but the cost of the eigen value calculation, i.e.,

Alex Slides critique all the other, taxonomy of the algorithms showing how they relate to the model, the cost model being comprehensive. algorithm does not capture the transfer cost.

move the below to appropriate places in this document looking at the mapping papers and tami tamur's paper on rebalancing ——————————————— Partitioning Sparse Matrices with Eigenvectors of Graphs - A heuristic algorithm is designed to compute a vertex separator in a general graph by first computing an edge separator in the graph from an eigenvector of the Laplacian matrix, and then using a maximum matching in a subgraph to compute the vertex separator.

we could describe the comm problem as an equilibrium problem, load balance is another equilibrium, mapping is equilibrium.; consider the nature of the tasks in terms of compute intensive vs comm intensive.

References

[1] Eitan Altman, Hisao Kameda, and Yoshihisa Hosokawa. "Nash equilibria in load balancing in distributed computer systems". In: *International Game Theory Review* 04.02 (2002), pp. 91–100. DOI: 10.1142/S0219198902000574. eprint: http://www.worldscientific.com/doi/pdf/10.1142/S0219198902000574. URL: http://www.worldscientific.com/doi/abs/10.1142/S0219198902000574.
The use of game theoretical techniques has been quite successful in describing routing in networks, both in road traffic applications as well as in telecommunication networks applications. We study in this paper a third area of applications of such games, which is load balancing in distributed computer systems. One of the most important questions that arise in all applications of routing games is the existence and uniqueness of equilibrium. Whereas the existence of Nash equilibrium is known for general models of networks under weak assumptions, uniqueness results are only known for very special applications, i.e., either for very special cost functions or for very special topologies. We establish in this paper the uniqueness of an equilibrium for routing games with topologies that model well distributed computer systems, under quite general assumptions on the costs.

[2] Cedric Augonnet et al. "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures". In: *Concurrency and Computation: Practice and Experience* 23.2 (2011), pp. 187–198. ISSN: 1532-0634. DOI: 10.1002/cpe.1631. URL: http://dx.doi.org/10.1002/cpe.1631.

In the field of HPC, the current hardware trend is to design multiprocessor architectures featuring heterogeneous technologies such as specialized coprocessors (e.g. Cell/BE) or data-parallel accelerators (e.g. GPUs). Approaching the theoretical performance of these architectures is a complex issue. Indeed, substantial efforts have already been devoted to efficiently offload parts of the computations. However, designing an execution model that unifies all computing units and associated embedded memory remains a main challenge. We therefore designed StarPU, an original runtime system providing a high-level, unified execution model tightly coupled with an expressive data management library. The main goal of StarPU is to provide numerical kernel designers with a convenient way to generate parallel tasks over heterogeneous hardware on the one hand, and easily develop and tune powerful scheduling algorithms on the other hand. We have developed several strategies that can be selected seamlessly at run-time, and we have analyzed their efficiency on several algorithms running simultaneously over multiple cores and a GPU. In addition to substantial improvements regarding execution times, we have obtained consistent superlinear parallelism by actually exploiting the heterogeneous nature of the machine. We eventually show that our dynamic approach competes with the highly optimized MAGMA library and overcomes the limitations of the corresponding static scheduling in a portable way.

[3] Seonmyeong Bak et al. "Handling Transient and Persistent Imbalance Together in Distributed and Shared Memory". In: *2018 IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. 2018.
The recent trend of rapidly increasing numbers of cores per chip has resulted in vast amounts of on-node parallelism. These high core counts result in hardware variability that introduces imbalance. Applications are also becoming more complex, resulting in dynamic load imbalance. Load imbalance of any kind can result in loss of performance and decrease in system utilization. We address the challenge of handling both transient and persistent load imbalances while maintaining locality and incurring low overhead. In this paper, we propose an integrated runtime system that combines the Charm++ distributed programming model with concurrent tasks to mitigate load imbalances within and across shared memory address spaces. It utilizes an infrequent periodic assignment of work to cores based on load measurement, in combination with user created tasks to handle load imbalance. We integrate OpenMP with Charm++ to enable creation of potential tasks via OpenMP's parallel loop construct. This is not specific to Charm++ and is also available to MPI applications through the Adaptive MPI implementation. We demonstrate the benefits of this integrated runtime system on three different applications. We show improvement of Lassen around 29.6% on Cori and 46.5% on Theta. We also demonstrate the benefits on a Charm++application, ChaNGa, by 25.7% on Theta, as well as an MPI proxy application, Kripke, using Adaptive MPI.

[4] Guy Baram and Tami Tamir. "Reoptimization of the minimum total flow-time scheduling problem". In: *Sustainable Computing: Informatics and Systems* 4.4 (2014). Special Issue on Energy Aware Resource Management and Scheduling (EARMS), pp. 241–251. ISSN: 2210-5379. DOI: https://doi.org/10.1016/j.suscom.2014.08.011. URL: http://www.sciencedirect.com/science/article/pii/S2210537914000523.
We consider reoptimization problems arising in dynamic scheduling environments, such as manufacturing systems and virtual machine managers. Due to changes in the environment (out-of-order or new resources, modified jobs processing requirements, etc.), the schedule needs to be modified. That is, jobs might be migrated from their current machine to a different one. Migrations are associated with a cost - due to relocation overhead and machine set-up times. In some systems, a migration is also associated with job extension. The goal is to find a good modified schedule, with a low transition cost from the initial one. We consider the objective of minimizing the total flow-time. We present optimal algorithms for the problem of achieving an optimal solution using the minimal possible transition cost. The algorithms and their running times depend on our assumptions on the instance and the allowed modifications. For the modification of machines addition, we also present an optimal algorithm for achieving the best possible schedule using a given limited budget for the transition.

[5] Rémi Barat. "Load balancing of multiphysics simulations by multi-criteria graph partitioning". Theses. Université de Bordeaux, Dec. 2017. URL: https://tel.archives-ouvertes.fr/tel-01672546.
Multiphysics simulation couple several computation phases. When they are run in parallel on memory-distributed architectures, minimizing the simulation time requires in most cases to balance the workload across computation units, for each computation phase. Moreover, the data distribution must minimize the induced communication. This problem can be modeled as a multi-criteria graph partitioning problem. We associate with each vertex of the graph a vector of weights, whose components, called "criteria", model the workload of the vertex for

each computation phase. The edges between vertices indicate data dependencies, and can be given a weight representing the communication volume transferred between the two vertices. The goal is to find a partition of the vertices that both balances the weights of each part for each criterion, and minimizes the "edgecut", that is, the sum of the weights of the edges cut by the partition. The maximum allowed imbalance is provided by the user, and we search for a partition that minimizes the edgecut, among all the partitions whose imbalance for each criterion is smaller than this threshold. This problem being NP-Hard in the general case, this thesis aims at devising and implementing heuristics that allow us to compute efficiently such partitions. Indeed, existing tools often return partitions whose imbalance is higher than the prescribed tolerance. Our study of the solution space, that is, the set of all the partitions respecting the balance constraints, reveals that, in practice, this space is extremely large. Moreover, we prove in the mono-criterion case that a bound on the normalized vertex weights guarantees the existence of a solution, and the connectivity of the solution space. Based on these theoretical results, we propose improvements of the multilevel algorithm. Existing tools implement many variations of this algorithm. By studying their source code, we emphasize these variations and their consequences, in light of our analysis of the solution space. Furthermore, we define and implement two initial partitioning algorithms, focusing on returning a solution. From a potentially imbalanced partition, they successively move vertices from one part to another. The first algorithm performs any move that reduces the imbalance, while the second performs at each step the move reducing the most the imbalance. We present an original data structure that allows us to optimize the choice of the vertex to move, and leads to partitions of imbalance smaller on average than existing methods. We describe the experimentation framework, named Crack, that we implemented in order to compare the various algorithms at stake. This comparison is performed by partitioning a set of instances including an industrial test case, and several fictitious cases. We define a method for generating realistic weight distributions corresponding to "Particles-in-Cells"-like simulations. Our results demonstrate the necessity to coerce the vertex weights during the coarsening phase of the multilevel algorithm. Moreover, we evidence the impact of the vertex ordering, which should depend on the graph topology, on the efficiency of the "Heavy-Edge" matching scheme. The various algorithms that we consider are implemented in an open-source graph partitioning software called Scotch. In our experiments, Scotch and Crack returned a balanced partition for each execution, whereas MeTiS, the current most used partitioning tool, fails regularly. Additionally, the edgecut of the solutions returned by Scotch and Crack is equivalent or better than the edgecut of the solutions returned by MeTiS.

[6] Sofia Belikovetsky and Tami Tamir. "Load rebalancing games in dynamic systems with migration costs". In: *Theoretical Computer Science* 622 (2016), pp. 16–33. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2016.01.030. URL: http://www.sciencedirect.com/science/article/pii/S0304397516000542.

We consider the following dynamic load balancing game: Given an initial assignment of jobs to identical parallel machines, the system is modified; specifically, some machines are added or removed. Each job's cost is the load on the machine it is assigned to; thus, when machines are added, jobs have an incentive to migrate to the new unloaded machines. When machines are removed, the jobs assigned to them must be reassigned. Consequently, other jobs might also benefit from migrations. In our job-extension penalty model, for a given extension parameter $\sigma \geq 0$, if the machine on which a job is assigned to in the modified schedule is different from its initial machine, then the jobs processing time is extended by σ. We provide answers to the basic questions arising in this model. Namely, the existence and calculation of a Nash Equilibrium and a Strong Nash Equilibrium, and their inefficiency compared to an optimal schedule. Our results show that the existence of job-migration penalties might lead to poor stable schedules; however, if the modification is a result of a sequence of improvement steps or, better, if the sequence of improvement steps can be supervised in some way (by forcing the jobs to play in a specific order) then any stable modified schedule approximates well an optimal one. Our work adds two realistic considerations to the study of job scheduling games: the analysis of the common situation in which systems are upgraded or suffer from failures, and the practical fact according to which job migrations are associated with a cost.

[7] P. Berenbrink et al. "Tight Load Balancing Via Randomized Local Search". In: *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. May 2017, pp. 192–201. DOI: 10.1109/IPDPS.2017.52.

We consider the following balls-into-bins process with n bins and m balls: Each ball is equipped with a mutually independent exponential clock of rate 1. Whenever a ball's clock rings, the ball samples a random bin and moves

there if the number of balls in the sampled bin is smaller than in its current bin. This simple process models a typical load balancing problem where users (balls) seek a selfish improvement of their assignment to resources (bins). From a game theoretic perspective, this is a randomized approach to the well-known KP model [1], while it is known as Randomized Local Search (RLS) in load balancing literature [2], [3]. Up to now, the best bound on the expected time to reach perfect balance was O((ln n)2+ln(n).n2/m) due to [3]. We improve this to an asymptotically tight O(ln(n)+n2/m). Our analysis is based on the crucial observation that performing destructive moves (reversals of RLS moves) cannot decrease the balancing time. This allows us to simplify problem instances and to ignore "inconvenient moves" in the analysis.

[8] Joanna Berlinska and Maciej Drozdowski. "Comparing load-balancing algorithms for MapReduce under Zipfian data skews". In: *Parallel Computing* 72 (2018), pp. 14–28. ISSN: 0167-8191. DOI: https://doi.org/10.1016/j.parco.2017.12.003. URL: http://www.sciencedirect.com/science/article/pii/S0167819117302065.

In this paper, we analyze applicability of various load-balancing methods in countering data skew in MapReduce computations. A MapReduce job consists of several phases: mapping, shuffling data, sorting and reducing. The distribution of the work in the last three phases is data-driven, and unequal distribution of the data keys may cause imbalance in the computation completion times and prolonged execution of the whole job. We propose algorithms of four different types for balancing computational effort in reduce-heavy MapReduce jobs and evaluate their performance under various degrees of data skew and system parameters. By applying an innovative method of visualizing algorithm dominance conditions, we are able to determine conditions under which certain load-balancing algorithms are capable of scheduling MapReduce computations well. We conclude that no single algorithm is a panacea and hybrid approaches are necessary.

[9] Muhammad Khurram Bhatti et al. "Locality-aware task scheduling for homogeneous parallel computing systems". In: *Computing* (Nov. 2017). ISSN: 1436-5057. DOI: 10.1007/s00607-017-0581-6. URL: https://doi.org/10.1007/s00607-017-0581-6.

In systems with complex many-core cache hierarchy, exploiting data locality can significantly reduce execution time and energy consumption of parallel applications. Locality can be exploited at various hardware and software layers. For instance, by implementing private and shared caches in a multi-level fashion, recent hardware designs are already optimised for locality. However, this would all be useless if the software scheduling does not cast the execution in a manner that promotes locality available in the programs themselves. Since programs for parallel systems consist of tasks executed simultaneously, task scheduling becomes crucial for the performance in multi-level cache architectures. This paper presents a heuristic algorithm for homogeneous multi-core systems called locality-aware task scheduling (LeTS). The LeTS heuristic is a work-conserving algorithm that takes into account both locality and load balancing in order to reduce the execution time of target applications. The working principle of LeTS is based on two distinctive phases, namely; working task group formation phase (WTG-FP) and working task group ordering phase (WTG-OP). The WTG-FP forms groups of tasks in order to capture data reuse across tasks while the WTG-OP determines an optimal order of execution for task groups that minimizes the reuse distance of shared data between tasks. We have performed experiments using randomly generated task graphs by varying three major performance parameters, namely: (1) communication to computation ratio (CCR) between 0.1 and 1.0, (2) application size, i.e., task graphs comprising of 50-, 100-, and 300-tasks per graph, and (3) number of cores with 2-, 4-, 8-, and 16-cores execution scenarios. We have also performed experiments using selected real-world applications. The LeTS heuristic reduces overall execution time of applications by exploiting inter-task data locality. Results show that LeTS outperforms state-of-the-art algorithms in amortizing inter-task communication cost.

[10] Robert D. Blumofe and Charles E. Leiserson. "Scheduling Multithreaded Computations by Work Stealing". In: *J. ACM* 46.5 (Sept. 1999), pp. 720–748. ISSN: 0004-5411. DOI: 10.1145/324133.324234. URL: http://doi.acm.org/10.1145/324133.324234.

This paper studies the problem of efficiently schedulling fully strict (i.e., well-structured) multithreaded computations on parallel computers. A popular and practical method of scheduling this kind of dynamic MIMD-style computation is work stealing, in which processors needing work steal computational threads from other processors. In this paper, we give the first provably good work-stealing scheduler for multithreaded computations with dependencies.Specifically, our analysis shows that the expected time to execute a fully strict computation on P processors using our work-stealing scheduler is T1/P + O(T inf , where T1 is the minimum serial execution time of the multithreaded computation and (T inf is the minimum execution time with an infinite number

of processors. Moreover, the space required by the execution is at most S1P, where S1 is the minimum serial space requirement. We also show that the expected total communication of the algorithm is at most O(PT inf (1 + nd)Smax), where Smax is the size of the largest activation record of any thread and nd is the maximum number of times that any thread synchronizes with its parent. This communication bound justifies the folk wisdom that work-stealing schedulers are more communication efficient than their work-sharing counterparts. All three of these bounds are existentially optimal to within a constant factor.

[11] Robert D. Blumofe et al. *Cilk: An Efficient Multithreaded Runtime System*. 1995.
Cilk (pronounced "silk") is a C-based runtime system for multithreaded parallel programming. In this paper, we document the efficiency of the Cilk work-stealing scheduler, both empirically and analytically. We show that on real and synthetic applications, the "work" and "critical-path length" of a Cilk computation can be used to model performance accurately. Consequently, a Cilk programmer can focus on reducing the computations work and critical-path length, insulated from load balancing and other runtime scheduling issues. We also prove that for the class of "fully stric"t (well-structured) programs, the Cilk scheduler achieves space, time, and communication bounds all within a constant factor of optimal. The Cilk runtime system currently runs on the Connection Machine CM5 MPP, the Intel Paragon MPP, the Sun Sparcstation SMP, and the Cilk-NOW network of workstations. Applications written in Cilk include protein folding, graphic rendering, backtrack search, and the Socrates chess program, which won second prize in the 1995 World Computer Chess Championship.

[12] J. E. Boillat. "Load Balancing and Poisson Equation in a Graph". In: *Concurrency: Pract. Exper.* 2.4 (Nov. 1990), pp. 289–313. ISSN: 1040-3108. DOI: 10.1002/cpe.4330020403. URL: http://dx.doi.org/10.1002/cpe.4330020403.
We present a fully distributed dynamic load balancing algorithm for parallel MIMD architectures. The algorithm can be described as a system of identical parallel processes, each running on a processor of an arbitrary interconnected network of processors. We show that the algorithm can be interpreted as a Poisson (heath) equation in a graph. This equation is analysed using Markov chain techniques and is proved to converge in polynomial time resulting in a global load balance. We also discuss some important parallel architectures and interconnection schemes such as linear processor arrays, tori, hypercubes, etc. Finally we present two applications where the algorithm has been successfully embedded (process mapping and molecular dynamic simulation).

[13] F. Busato and N. Bombieri. "A performance, power, and energy efficiency analysis of load balancing techniques for GPUs". In: *2017 12th IEEE International Symposium on Industrial Embedded Systems (SIES)*. June 2017, pp. 1–8. DOI: 10.1109/SIES.2017.7993387.
Load balancing is a key aspect to face when implementing any parallel application for Graphic Processing Units (GPUs). It is particularly crucial if one considers that it strongly impacts on performance, power and energy efficiency of the whole application. Many different partitioning techniques have been proposed in the past to deal with either very regular workloads (static techniques) or with irregular workloads (dynamic techniques). Nevertheless, it has been proven that no one of them provides a sound trade-off, from the performance point of view, when applied in both cases. More recently, a dynamic multi-phase approach has been proposed for workload partitioning and work item-to-thread allocation. Thanks to its very low complexity and several architecture-oriented optimizations, it can provide the best results in terms of performance with respect to the other approaches in the literature with both regular and irregular datasets. Besides the performance comparison, no analysis has been conducted to show the effect of all these techniques on power and energy consumption on both GPUs for desktop and GPUs for low-power embedded systems. This paper shows and compares, in terms of performance, power, and energy efficiency, the experimental results obtained by applying all the different static, dynamic, and semi-dynamic techniques at the state of the art to different datasets and over different GPU technologies (i.e., NVIDIA Maxwell GTX 980 device, NVIDIA Jetson Kepler TK1 low-power embedded system).

[14] Luis Miguel Campos and Isaac D Scherson. "Rate of change load balancing in distributed and parallel systems". In: *Parallel Computing* 26.9 (2000), pp. 1213–1230. ISSN: 0167-8191. DOI: https://doi.org/10.1016/S0167-8191(00)00036-3. URL: http://www.sciencedirect.com/science/article/pii/S0167819100000363.
Dynamic load balancing (DLB) is an important system function destined to distribute workload among available processors to improve throughput and/or execution times of parallel computer programs either uniform or non-uniform (jobs whose workload varies at run-time in unpredictable ways). Non-uniform computation and

communication requirements may bog down a parallel computer if no efficient load distribution is effected. A novel distributed algorithm for load balancing is proposed and is based on local rate of change (RoC) observations rather than on global absolute load numbers. It is a totally distributed algorithm and requires no centralized trigger and/or decision makers. The strategy is discussed and analyzed by means of experimental simulation.

[15] U. V. Catalyurek et al. "Hypergraph-based Dynamic Load Balancing for Adaptive Scientific Computations". In: *2007 IEEE International Parallel and Distributed Processing Symposium*. Mar. 2007, pp. 1–11. DOI: 10.1109/IPDPS.2007.370258.
Adaptive scientific computations require that periodic repartitioning (load balancing) occur dynamically to maintain load balance. Hypergraph partitioning is a successful model for minimizing communication volume in scientific computations, and partitioning software for the static case is widely available. In this paper, we present a new hypergraph model for the dynamic case, where we minimize the sum of communication in the application plus the migration cost to move data, thereby reducing total execution time. The new model can be solved using hypergraph partitioning with faced vertices. We describe an implementation of a parallel multilevel repartitioning algorithm within the Zoltan load-balancing toolkit, which to our knowledge is the first code for dynamic load balancing based on hypergraph partitioning. Finally, we present experimental results that demonstrate the effectiveness of our approach on a Linux cluster with up to 64 processors. Our new algorithm compares favorably to the widely used ParMETIS partitioning software in terms of quality, and would have reduced total execution time in most of our test cases.

[16] Daniel Cederman and Philippas Tsigas. "On Dynamic Load Balancing on Graphics Processors". In: *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*. GH '08. Sarajevo, Bosnia and Herzegovina: Eurographics Association, 2008, pp. 57–64. ISBN: 978-3-905674-09-5. URL: http://dl.acm.org/citation.cfm?id=1413957.1413967.
To get maximum performance on the many-core graphics processors it is important to have an even balance of the workload so that all processing units contribute equally to the task at hand. This can be hard to achieve when the cost of a task is not known beforehand and when new sub-tasks are created dynamically during execution. With the recent advent of scatter operations and atomic hardware primitives it is now possible to bring some of the more elaborate dynamic load balancing schemes from the conventional SMP systems domain to the graphics processor domain. We have compared four different dynamic load balancing methods to see which one is most suited to the highly parallel world of graphics processors. Three of these methods were lock-free and one was lock-based. We evaluated them on the task of creating an octree partitioning of a set of particles. The experiments showed that synchronization can be very expensive and that new methods that take more advantage of the graphics processors features and capabilities might be required. They also showed that lock-free methods achieves better performance than blocking and that they can be made to scale with increased numbers of processing units.

[17] Quan Chen and Minyi Guo. "Locality-Aware Work Stealing Based on Online Profiling and Auto-Tuning for Multisocket Multicore Architectures". In: *ACM Trans. Archit. Code Optim.* 12.2 (July 2015), 22:1–22:24. ISSN: 1544-3566. DOI: 10.1145/2766450. URL: http://doi.acm.org/10.1145/2766450.
Modern mainstream powerful computers adopt multisocket multicore CPU architecture and NUMA-based memory architecture. While traditional work-stealing schedulers are designed for single-socket architectures, they incur severe shared cache misses and remote memory accesses in these computers. To solve the problem, we propose a locality-aware work-stealing (LAWS) scheduler, which better utilizes both the shared cache and the memory system. In LAWS, a load-balanced task allocator is used to evenly split and store the dataset of a program to all the memory nodes and allocate a task to the socket where the local memory node stores its data for reducing remote memory accesses. Then, an adaptive DAG packer adopts an auto-tuning approach to optimally pack an execution DAG into cache-friendly subtrees. After cache-friendly subtrees are created, every socket executes cache-friendly subtrees sequentially for optimizing shared cache usage. Meanwhile, a triple-level work-stealing scheduler is applied to schedule the subtrees and the tasks in each subtree. Through theoretical analysis, we show that LAWS has comparable time and space bounds compared with traditional work-stealing schedulers. Experimental results show that LAWS can improve the performance of memory-bound programs.

[18] George Cybenko. "Dynamic load balancing for distributed memory multiprocessors". In: *Journal of Parallel and Distributed Computing* 7.2 (1989), pp. 279–301. ISSN: 0743-7315. DOI: https://doi.org/

10.1016/0743-7315(89)90021-X. URL: http://www.sciencedirect.com/science/article/pii/074373158990021X.
In this paper we study diffusion schemes for dynamic load balancing on message passing multiprocessor networks. One of the main results concerns conditions under which these dynamic schemes converge and their rates of convergence for arbitrary topologies. These results use the eigenstructure of the iteration matrices that arise in dynamic load balancing. We completely analyze the hypercube network by explicitly computing the eigenstructure of its node adjacency matrix. Using a realistic model of inter-processor communications, we show that a diffusion approach to load balancing on a hypercube multiprocessor is inferior to another approach which we call the dimension exchange method. For a d-dimensional hypercube, we compute the rate of convergence to a uniform work distribution and show that after d + 1 iterations of a diffusion type approach, we can guarantee that the work distribution is approximately within e - 2 of the uniform distribution independent of the hypercube dimension d. Both static and dynamic random models of work distribution are studied.

[19] A. Danalis et al. "PaRSEC in Practice: Optimizing a Legacy Chemistry Application through Distributed Task-Based Execution". In: *2015 IEEE International Conference on Cluster Computing*. Sept. 2015, pp. 304–313. DOI: 10.1109/CLUSTER.2015.50.
Task-based execution has been growing in popularity as a means to deliver a good balance between performance and portability in the post-petascale era. The Parallel Runtime Scheduling and Execution Control (PARSEC) framework is a task-based runtime system that we designed to achieve high performance computing at scale. PARSEC offers a programming paradigm that is different than what has been traditionally used to develop large scale parallel scientific applications. In this paper, we discuss the use of PARSEC to convert a part of the Coupled Cluster (CC) component of the Quantum Chemistry package NWCHEM into a task-based form. We explain how we organized the computation of the CC methods in individual tasks with explicitly defined data dependencies between them and re-integrated the modified code into NWCHEM. We present a thorough performance evaluation and demonstrate that the modified code outperforms the original by more than a factor of two. We also compare the performance of different variants of the modified code and explain the different behaviors that lead to the differences in performance.

[20] Yunhua Deng and Rynson W. H. Lau. "Heat Diffusion Based Dynamic Load Balancing for Distributed Virtual Environments". In: *Proceedings of the 17th ACM Symposium on Virtual Reality Software and Technology*. VRST '10. Hong Kong: ACM, 2010, pp. 203–210. ISBN: 978-1-4503-0441-2. DOI: 10.1145/1889863.1889910. URL: http://doi.acm.org/10.1145/1889863.1889910.
Distributed virtual environments (DVEs) are becoming very popular in recent years, due to their application in online gaming and social networking. One of the main research problems in DVEs is on how to balance the workload when a lot of concurrent users are accessing it. There are a number of load balancing methods proposed to address this problem. However, they either spend too much time on optimizing the partitioning process and become too slow or emphasize on efficiency and the repartitioning process becomes too ineffective. In this paper, we propose a new dynamic load balancing approach for DVEs based on the heat diffusion approach which has been studied in other areas and proved to be very effective and efficient for dynamic load balancing. We have two main contributions. First, we propose an efficient cell selection scheme to identify and select appropriate cells for load migration. Second, we propose two heat diffusion based load balancing algorithms, local and global diffusion. Our results show that the new algorithms are both efficient and effective compared with some existing methods, and the global diffusion method performs the best.

[21] Karen D. Devine et al. "New challenges in dynamic load balancing". In: *Applied Numerical Mathematics* 52.2 (2005). ADAPT '03: Conference on Adaptive Methods for Partial Differential Equations and Large-Scale Computation, pp. 133–152. ISSN: 0168-9274. DOI: https://doi.org/10.1016/j.apnum.2004.08.028. URL: http://www.sciencedirect.com/science/article/pii/S0168927404001631.
Data partitioning and load balancing are important components of parallel computations. Many different partitioning strategies have been developed, with great effectiveness in parallel applications. But the load-balancing problem is not yet solved completely; new applications and architectures require new partitioning features. Existing algorithms must be enhanced to support more complex applications. New models are needed for non-square, non-symmetric, and highly connected systems arising from applications in biology, circuits, and materials simulations. Increased use of heterogeneous computing architectures requires partitioners that account for non-uniform computing, network, and memory resources. And, for greatest impact, these new capabilities

must be delivered in toolkits that are robust, easy-to-use, and applicable to a wide range of applications. In this paper, we discuss our approaches to addressing these issues within the Zoltan Parallel Data Services toolkit.

[22] Gerrett Diamond, Cameron W. Smith, and Mark S. Shephard. "Dynamic Load Balancing of Massively Parallel Unstructured Meshes". In: *Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*. ScalA '17. Denver, Colorado: ACM, 2017, 9:1–9:7. ISBN: 978-1-4503-5125-6. DOI: 10.1145/3148226.3148236. URL: http://doi.acm.org/10.1145/3148226.3148236.

Simulating systems with evolving relational structures on massively parallel computers require the computational work to be evenly distributed across the processing resources throughout the simulation. Adaptive, unstructured, mesh-based finite element and finite volume tools best exemplify this need. We present EnGPar and its diffusive partition improvement method that accounts for multiple application specified criteria. EnGPar performance is compared against its predecessor, ParMA. Specifically, partition improvement results are provided on up to 512Ki processes of the Argonne Leadership Computing Facility Mira BlueGene/Q system.

[23] Goran Flegar and Hartwig Anzt. "Overcoming Load Imbalance for Irregular Sparse Matrices". In: *Proceedings of the Seventh Workshop on Irregular Applications: Architectures and Algorithms*. IA3'17. Denver, CO, USA: ACM, 2017, 2:1–2:8. ISBN: 978-1-4503-5136-2. DOI: 10.1145/3149704.3149767. URL: http://doi.acm.org/10.1145/3149704.3149767.

In this paper we propose a load-balanced GPU kernel for computing the sparse matrix vector (SpMV) product. Making heavy use of the latest GPU programming features, we also enable satisfying performance for irregular and unbalanced matrices. In a performance comparison using 400 test matrices we reveal the new kernel being superior to the most popular SpMV implementations.

[24] Juan J. Galvez, Nikhil Jain, and Laxmikant V. Kale. "Automatic Topology Mapping of Diverse Large-scale Parallel Applications". In: *Proceedings of the International Conference on Supercomputing*. ICS '17. Chicago, Illinois: ACM, 2017, 17:1–17:10. ISBN: 978-1-4503-5020-4. DOI: 10.1145/3079079.3079104. URL: http://doi.acm.org/10.1145/3079079.3079104.

Topology-aware mapping aims at assigning tasks to processors in a way that minimizes network load, thus reducing the time spent waiting for communication to complete. Many mapping schemes and algorithms have been proposed. Some are application or domain specific, and others require significant effort by developers or users to successfully apply them. Moreover, a task mapping algorithm by itself is not enough to map the diverse set of applications that exist. Applications can have distinct communication patterns, from point-to-point communication with neighbors in a virtual process grid, to irregular point-to-point communication, to different types of collectives with differing group sizes, and any combination of the above. These patterns should be analyzed, and critical patterns extracted and automatically provided to the mapping algorithm, all without specialized user input. To our knowledge, this problem has not been addressed before for the general case. In this paper, we propose a complete and automatic mapping system that does not require special user involvement, works with any application, and whose mapping performs better than existing schemes, for a wide range of communication patterns and machine topologies. This makes it suitable for online mapping of HPC applications in many different scenarios. We evaluate our scheme with several applications exhibiting different communication patterns (including collectives) on machines with 3D torus, 5D torus and fat-tree network topologies, and show up to 2.2x performance improvements.

[25] Tianhan Gao, Xiaoyu Luo, and Nan Guo. "Multi-frame Prediction Load Balancing Algorithm for Sort-first Parallel Rendering". In: *Proceedings of the 2017 International Conference on Computer Graphics and Digital Image Processing*. CGDIP '17. Prague, Czech Republic: ACM, 2017, 3:1–3:5. ISBN: 978-1-4503-5236-9. DOI: 10.1145/3110224.3110240. URL: http://doi.acm.org/10.1145/3110224.3110240.

In order to solve the problem of the load unbalance in sort-first parallel rendering system, this paper proposes a multi-frame prediction algorithm derived from CSLB algorithm to improve performence in parallel rendering, which introduces multi-frame feedback prediction to achieve better practical results. The experiment is carried out on the Equalizer parallel rendering framework and the results show that the proposed algorithm can improve the system frame rate well compared with the CSLB algorithm in terms of sort-first parallel rendering.

[26] Daniel Grosu and Anthony T. Chronopoulos. "Noncooperative load balancing in distributed systems". In: *Journal of Parallel and Distributed Computing* 65.9 (2005), pp. 1022–1034. ISSN: 0743-7315. DOI:

https://doi.org/10.1016/j.jpdc.2005.05.001. URL: http://www.sciencedirect.com/science/article/pii/S0743731505001152.
In this paper, we present a game theoretic framework for obtaining a user-optimal load balancing scheme in heterogeneous distributed systems. We formulate the static load balancing problem in heterogeneous distributed systems as a noncooperative game among users. For the proposed noncooperative load balancing game, we present the structure of the Nash equilibrium. Based on this structure we derive a new distributed load balancing algorithm. Finally, the performance of our noncooperative load balancing scheme is compared with that of other existing schemes. The main advantages of our load balancing scheme are the distributed structure, low complexity and optimality of allocation for each user.

[27] A. Heirich. "A Scalable Diffusion Algorithm for Dynamic Mapping and Load Balancing on Networks of Arbitrary Topology". In: *International Journal of Foundations of Computer Science* 08.03 (1997), pp. 329–346. DOI: 10.1142/S0129054197000215. eprint: http://www.worldscientific.com/doi/pdf/10.1142/S0129054197000215. URL: http://www.worldscientific.com/doi/abs/10.1142/S0129054197000215.
The problems of mapping and load balancing applications on arbitrary networks are considered. A novel diffusion algorithm is presented to solve the mapping problem. It complements the well known diffusive algorithms for load balancing which have enjoyed success on massivel parallel computers (MPPs). Mapping is more difficult on interconnection networks than on MPPs because of the variations that occur in network topology. Popular mapping algorithms for MPPs which depend on recursive topologies are not applicable to irregular networks. The most celebrated of these MPP algorithms use information from the Laplacian matrix of a graph of communicating processes. The diffusion algorithm presented in this paper is also derived from this Laplacian matrix. The diffusion algorithm works on arbitrary network topologies and is dramatically faster than the celebrated MPP algorithms. It is delay and fault tolerant. Time to convergence depends on initial conditions and is insensitive to problem scale. This excellent scalability, among other features, makes the diffusion algorithm a viable candidate for dynamically mapping and load balancing not only existing MPP systems but also large distributed systems like the Internet, small cluster computers, and networks of workstations.

[28] Alan Heirich and Stephen Taylor. *A Parabolic Load Balancing Method*. Tech. rep. 1994. DOI: 10.13140/RG.2.2.17432.08966.
This paper presents a diffusive load balancing methods for scalable multicomputers. In contrast to other schemes which are provably correct the method scales to large numbers of processors with no increase in run time. In contrast to other schemes which are scalable the method is provably correct and the paper analyses the rate of convergence. To control aggregate cpu idle time it can be useful to balance the load to specifiable accuracy. The method achieves arbitrary accuracy by proper consideraton of numerical error and stability. This paper presents the method, proves correctness, convergence and scalability, and simulates applications to generic problems in computational fluid dynamics (CFD). The applications reveal some useful properties. The method can preserve adjacency relationships among elements of an adapting computational domain. This make it useful for partitioning unstructured computational grids in concurrent computations. The method can execute asynchronously to balance a subportion of a domain without affecting the rest of the domain. Theory and experiment show the method is efficient on the scalable multicomputers of the present and coming years. The number of floating point operations required per processor to reduce a point disturbance by 90% is 168 on a system of 512 computers and 105 on a system of 1,000,000 computers. On a typical contemporary multicomputer [19] this requires 82.5 μs of wall-clock time.

[29] G Horton. "A multi-level diffusion method for dynamic load balancing". In: *Parallel Computing* 19.2 (1993), pp. 209–218. ISSN: 0167-8191. DOI: https://doi.org/10.1016/0167-8191(93)90050-U. URL: http://www.sciencedirect.com/science/article/pii/016781919390050U.
We consider the problem of dynamic load balancing for multiprocessors, for which a typical application is a parallel finite element solution method using non-structured grids and adaptive grid refinement. This type of application requires communication between the subproblems which arises from the interdependencies in the data. A load balancing algorithm should ideally not make any assumptions about the physical topology of the parallel machine. Further requirements are that the procedure should be fast and accurate. An new multi-level algorithm is presented for solving the dynamic load balancing problem which has these properties and whose parallel complexity is logarithmic in the number of processors used in the computation.

[30] Matthias Lieber, Kerstin Goessner, and Wolfgang E. Nagel. "The Potential of Diffusive Load Balancing at Large Scale". In: *Proceedings of the 23rd European MPI Users' Group Meeting*. EuroMPI 2016. Edinburgh, United Kingdom: ACM, 2016, pp. 154–157. ISBN: 978-1-4503-4234-6. DOI: 10.1145/2966884.2966887. URL: http://doi.acm.org/10.1145/2966884.2966887.
Dynamic load balancing with diffusive methods is known to provide minimal load transfer and requires communication between neighbor nodes only. These are very attractive properties for highly parallel systems. We compare diffusive methods with state-of-the-art geometrical and graph-based partitioning methods on thousands of nodes. When load balancing overheads, i.e. repartitioning computation time and migration, have to be minimized, diffusive methods provide substantial benefits.

[31] Hui Liu et al. "Dynamic Load Balancing using Hilbert Space-filling Curves for Parallel Reser- voir Simulations". In: *Proc. SPE Reservoir Simulation Conference*. SPE'17. Montgomery, TX: Society for Petroleum Engineering, 2017.
New reservoir simulators designed for parallel computers enable us to overcome performance limitations of personal computers and to simulate large-scale reservoir models. With development of parallel reservoir simulators, more complex physics and detailed models can be studied. The key to design efficient parallel reservoir simulators is not to improve the performance of individual CPUs drastically but to utilize the aggregation of computing power of all requested nodes through high speed networks. An ideal scenario is that when the number of MPI (Message Passing Interface) processes is doubled, the running time of parallel reservoir simulators is reduced by half. The goal of load balancing (grid partitioning) is to minimize overall computations and communications, and to make sure that all processors have a similar workload. Geometric methods divide a grid by using a location of a cell while topological methods work with connectivity of cells, which is generally described as a graph. This paper introduces a Hilbert space-filling curve method. A space-filling curve is a continuous curve and defines a map between a one-dimensional space and a multi-dimensional space. A Hilbert space-filling curve is one special space-filling curve discovered by Hilbert and has many useful characteristics, such as good locality, which means that two objects that are close to each other in a multi-dimensional space are also close to each other in a one dimensional space. This property can model communications in grid-based parallel applications. The idea of the Hilbert space-filling curve method is to map a computational domain into a one-dimensional space, partition the one-dimensional space to certain intervals, and assign all cells in a same interval to a MPI process. To implement a load balancing method, a mapping kernel is required to convert high-dimensional coordinates to a scalar value and an efficient one-dimensional partitioning module that divides a one-dimensional space and makes sure that all intervals have a similar workload. The Hilbert space-filling curve method is compared with ParMETIS, a famous graph partitioning package. The results show that our Hilbert space-filling curve method has good partition quality. It has been applied to grids with billions of cells, and linear scalability has been obtained on IBM Blue Gene/Q.

[32] J. Maglalang, S. Krishnamoorthy, and K. Agrawal. "Locality-Aware Dynamic Task Graph Scheduling". In: *2017 46th International Conference on Parallel Processing (ICPP)*. Aug. 2017, pp. 70–80. DOI: 10.1109/ICPP.2017.16.
Dynamic task graph schedulers automatically balance work across processor cores by scheduling tasks among available threads while preserving dependences. In this paper, we design NABBITC, a provably efficient dynamic task graph scheduler that accounts for data locality on NUMA systems. NABBITC allows users to assign a color to each task representing the location (e.g., a processor core) that has the most efficient access to data needed during that node's execution. NABBITC then automatically adjusts the scheduling so as to preferentially execute each node at the location that matches its color-leading to better locality because the node is likely to make local rather than remote accesses. At the same time, NABBITC tries to optimize load balance and not add too much overhead compared to the vanilla NABBIT scheduler that does not consider locality. We provide a theoretical analysis that shows that NABBITC does not asymptotically impact the scalability of NABBIT.We evaluated the performance of NABBITC on a suite of benchmarks, including both memory and compute intensive applications. Our experiments indicate that adding locality awareness has a considerable performance advantage compared to the vanilla NABBIT scheduler. Furthermore, we compared NABBITC to both OpenMP tasks and OpenMP loops. For regular applications, OpenMP loops can achieve perfect locality and perfect load balance statically. For these benchmarks, NABBITC has a small performance penalty compared to OpenMP due to its dynamic scheduling strategy. Similarly, for compute intensive applications with course-grained tasks, OpenMP task's centralized scheduler provides the best performance. However, we

find that NABBITC provides a good trade-off between data locality and load balance; on memory intensive jobs, it consistently outperforms OpenMP tasks while for irregular jobs where load balancing is important, it outperforms OpenMP loops. Therefore, NABBITC combines the benefits of locality-aware- scheduling for regular, memory intensive, applications (the forte of static schedulers such as those in OpenMP) and dynamically adapting to load imbalance in irregular applications (the forte of dynamic schedulers such as Cilk Plus, TBB, and Nabbit).

[33] Harshitha Menon and Laxmikant Kalé. "A Distributed Dynamic Load Balancer for Iterative Applications". In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC '13. Denver, Colorado: ACM, 2013, 15:1–15:11. ISBN: 978-1-4503-2378-9. DOI: 10.1145/2503210.2503284. URL: http://doi.acm.org/10.1145/2503210.2503284.
For many applications, computation load varies over time. Such applications require dynamic load balancing to improve performance. Centralized load balancing schemes, which perform the load balancing decisions at a central location, are not scalable. In contrast, fully distributed strategies are scalable but typically do not produce a balanced work distribution as they tend to consider only local information. This paper describes a fully distributed algorithm for load balancing that uses partial information about the global state of the system to perform load balancing. This algorithm, referred to as GrapevineLB, consists of two stages: global information propagation using a lightweight algorithm inspired by epidemic [21] algorithms, and work unit transfer using a randomized algorithm. We provide analysis of the algorithm along with detailed simulation and performance comparison with other load balancing strategies. We demonstrate the effectiveness of GrapevineLB for adaptive mesh refinement and molecular dynamics on up to 131,072 cores of BlueGene/Q.

[34] Olga Pearce et al. "Exploring dynamic load imbalance solutions with the CoMD proxy application". In: *Future Generation Computer Systems* (2017). ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2017.12.010. URL: http://www.sciencedirect.com/science/article/pii/S0167739X17300560.
Proxy applications are developed to simplify studying parallel performance of scientific simulations and to test potential solutions for performance problems. However, proxy applications are typically too simple to allow work migration or to represent the load imbalance of their parent applications. To study the ability of load balancing solutions to balance work effectively, we enable work migration in one of the Exascale Co-design Center for Materials in Extreme Environments (ExMatEx) [1] applications, CoMD. We design a methodology to parameterize three key aspects necessary for studying load imbalance correction: (1) the granularity with which work can be migrated; (2) the initial load imbalance; (3) the dynamic load imbalance (how quickly the load changes over time). We present a study of the impact of flexibility in work migration in CoMD on load balance and the associated rebalancing costs for a wide range of initial and dynamic load imbalance scenarios.

[35] Ali Pinar and Cevdet Aykanat. "Fast optimal load balancing algorithms for 1D partitioning". In: *Journal of Parallel and Distributed Computing* 64.8 (2004), pp. 974–996. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2004.05.003. URL: http://www.sciencedirect.com/science/article/pii/S0743731504000851.
The one-dimensional decomposition of nonuniform workload arrays with optimal load balancing is investigated. The problem has been studied in the literature as the "chains-on-chains partitioning" problem. Despite the rich literature on exact algorithms, heuristics are still used in parallel computing community with the hope of good decompositions and the myth of exact algorithms being hard to implement and not runtime efficient. We show that exact algorithms yield significant improvements in load balance over heuristics with negligible overhead. Detailed pseudocodes of the proposed algorithms are provided for reproducibility. We start with a literature review and propose improvements and efficient implementation tips for these algorithms. We also introduce novel algorithms that are asymptotically and runtime efficient. Our experiments on sparse matrix and direct volume rendering datasets verify that balance can be significantly improved by using exact algorithms. The proposed exact algorithms are 100 times faster than a single sparse-matrix vector multiplication for 64-way decompositions on the average. We conclude that exact algorithms with proposed efficient implementations can effectively replace heuristics.

[36] J. Posner and C. Fohry. "Fault Tolerance for Cooperative Lifeline-Based Global Load Balancing in Java with APGAS and Hazelcast". In: *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. May 2017, pp. 854–863. DOI: 10.1109/IPDPSW.2017.31.

Fault tolerance is a major issue for parallel applications. Approaches on application-level are gaining increasing attention because they may be more efficient than system-level ones. In this paper, we present a generic reusable framework for fault-tolerant parallelization with the task pool pattern. Users of this framework can focus on coding sequential tasks for their problem, while respecting some framework contracts. The framework is written in Java and deploys the APGAS library as well as Hazelcasts distributed and fault-tolerant IMap. Our fault-tolerance scheme uses two system-wide maps, in which it stores, e.g., backups of local task pools. Framework users may configure the number of backup copies to control how many simultaneous failures are tolerated. The algorithm is correct in the sense that the computed result is the same as in non-failure case, or the program aborts with an error message. In experiments with up to 128 workers, we compared the frameworks performance with that of a non-fault-tolerant variant during failure-free operation. For the UTS and BC benchmarks, the overhead was at most 35%. Measured values were similar as for a related, but less flexible fault-tolerant X10 framework, without a clear winner. Raising the number of backup copies to six only marginally improved the overhead.

[37] Jonas Posner and Claudia Fohry. "A Java Task Pool Framework providing Fault-Tolerant Global Load Balancing". In: *International Journal of Networking and Computing* 8 (2018).
Fault tolerance is gaining importance in parallel computing, especially on large clusters. Traditional approaches handle the issue on system-level. Application-level approaches are becoming increasingly popular, since they may be more efficient. This paper presents a fault-tolerant work stealing technique on application level, and describes its implementation in a generic reusable task pool framework for Java. When using this framework, programmers can focus on writing sequential code to solve their actual problem. The framework is written in Java and utilizes the APGAS library for parallel programming. It implements a comparatively simple algorithm that relies on a resilient data structure for storing backups of local pools and other information. Our implementation uses Hazelcast's IMap for this purpose, which is an automatically distributed and fault-tolerant key-value store. The number of backup copies is configurable and determines how many simultaneous failures can be tolerated. Our algorithm is shown to be correct in the sense that failures are either tolerated and the computed result is the same as in non-failure case, or the program aborts with an error message. Experiments were conducted with the UTS, NQueens and BC benchmarks on up to 144 workers. First, we compared the performance of our framework with that of a non-fault-tolerant variant during failure-free operation. Depending on the parameter settings, the overhead was at most 46.06%. The particular value tended to increase with the number of steals. Second, we compared our framework's performance with that of a related, but less flexible fault-tolerant X10 framework. Here, we did not observe a clear winner. Finally, we measured the overheads for restoring a failed worker and for raising the number of backup copies from one to six, and found both to be negligible.

[38] Jonas Posner and Claudia Fohry. "Hybrid work stealing of locality-flexible and cancelable tasks for the APGAS library". In: *The Journal of Supercomputing* (Jan. 2018). ISSN: 1573-0484. DOI: 10.1007/s11227-018-2234-8. URL: https://doi.org/10.1007/s11227-018-2234-8.
Since large parallel machines are typically clusters of multicore nodes, parallel programs should be able to deal with both shared memory and distributed memory. This paper proposes a hybrid work stealing scheme, which combines the lifeline-based variant of distributed task pools with the node-internal load balancing of Java's Fork/Join framework. We implemented our scheme by extending the APGAS library for Java, which is a branch of the X10 project. APGAS programmers can now spawn locality-flexible tasks with a new asyncAny construct. These tasks are transparently mapped to any resource in the overall system, so that the load is balanced over both nodes and cores. Unprocessed asyncAny-tasks can also be cancelled. In performance measurements with up to 144 workers on up to 12 nodes, we observed near linear speedups for four benchmarks and a low overhead for cancellation-related bookkeeping.

[39] Alex Pothen, Horst D. Simon, and Kang-Pu Liou. "Partitioning Sparse Matrices with Eigenvectors of Graphs". In: *SIAM Journal on Matrix Analysis and Applications* 11.3 (1990), pp. 430–452. DOI: 10.1137/0611030. eprint: https://doi.org/10.1137/0611030. URL: https://doi.org/10.1137/0611030.
The problem of computing a small vertex separator in a graph arises in the context of computing a good ordering for the parallel factorization of sparse, symmetric matrices. An algebraic approach for computing vertex separators is considered in this paper. It is, shown that lower bounds on separator sizes can be obtained in terms of the eigenvalues of the Laplacian matrix associated with a graph. The Laplacian eigenvectors of

grid graphs can be computed from Kronecker products involving the eigenvectors of path graphs, and these eigenvectors can be used to compute good separators in grid graphs. A heuristic algorithm is designed to compute a vertex separator in a general graph by first computing an edge separator in the graph from an eigenvector of the Laplacian matrix, and then using a maximum matching in a subgraph to compute the vertex separator. Results on the quality of the separators computed by the spectral algorithm are presented, and these are compared with separators obtained from other algorithms for computing separators. Finally, the time required to compute the Laplacian eigenvector is reported, and the accuracy with which the eigenvector must be computed to obtain good separators is considered. The spectral algorithm has the advantage that it can be implemented on a medium-size multiprocessor in a straightforward manner.

[40] Ananya Raval et al. "Dynamic Load Balancing Strategies for Graph Applications on GPUs". In: *arXiv:1711.00231 [cs.DC]* (2017).
Acceleration of graph applications on GPUs has found large interest due to the ubiquitous use of graph processing in various domains. The inherent *irregularity* in graph applications leads to several challenges for parallelization. A key challenge, which we address in this paper, is that of load-imbalance. If the work-assignment to threads uses node-based graph partitioning, it can result in skewed task-distribution, leading to poor load-balance. In contrast, if the work-assignment uses edge-based graph partitioning, the load-balancing is better, but the memory requirement is relatively higher. This makes it unsuitable for large graphs. In this work, we propose three techniques for improved load-balancing of graph applications on GPUs. Each technique brings in unique advantages, and a user may have to employ a specific technique based on the requirement. Using Breadth First Search and Single Source Shortest Paths as our processing kernels, we illustrate the effectiveness of each of the proposed techniques in comparison to the existing node-based and edge-based mechanisms.

[41] Tiberiu Rotaru and Hans-Heinrich Nageli. "Dynamic load balancing by diffusion in heterogeneous systems". In: *Journal of Parallel and Distributed Computing* 64.4 (2004), pp. 481–497. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2004.02.001. URL: http://www.sciencedirect.com/science/article/pii/S0743731504000371.
The distributed environments constitute a major option for the future development of high-performance computing. In order to be able to efficiently execute parallel applications on such systems, one should ensure a fair utilization of the available resources. Here, we address a number of aspects regarding the generalization of the diffusion algorithms for the case when the processors have different relative speeds and the communication parameters have different values. Although some work has been done in this direction, we propose complementary results and we investigate other variants than those commonly used. In a first step, we discuss general aspects of the generalized diffusion. Bounds are formulated for the convergence factor and an explicit expression is given for the migration flow generated by such algorithms. It is shown that this flow has an important property, that is a scaled projection of all other balancing flows. In the second part, a variant of generalized diffusion is investigated. Complexity results are formulated and it is shown that this algorithm theoretically converges faster than the hydrodynamic algorithm. Comparative tests between different variants of generalized diffusion algorithms are performed.

[42] Vijay A. Saraswat et al. "Lifeline-based Global Load Balancing". In: *Proceedings of the 16th ACM Symposium on Principles and Practice of Parallel Programming*. PPoPP '11. San Antonio, TX, USA: ACM, 2011, pp. 201–212. ISBN: 978-1-4503-0119-0. DOI: 10.1145/1941553.1941582. URL: http://doi.acm.org/10.1145/1941553.1941582.
On shared-memory systems, Cilk-style work-stealing has been used to effectively parallelize irregular task-graph based applications such as Unbalanced Tree Search (UTS). There are two main difficulties in extending this approach to distributed memory. In the shared memory approach, thieves (nodes without work) constantly attempt to asynchronously steal work from randomly chosen victims until they find work. In distributed memory, thieves cannot autonomously steal work from a victim without disrupting its execution. When work is sparse, this results in performance degradation. In essence, a direct extension of traditional work-stealing to distributed memory violates the work-first principle underlying work-stealing. Further, thieves spend useless CPU cycles attacking victims that have no work, resulting in system inefficiencies in multi-programmed contexts. Second, it is non-trivial to detect active distributed termination (detect that programs at all nodes are looking for work, hence there is no work). This problem is well-studied and requires careful design for good performance. Unfortunately, in most existing languages/frameworks, application developers are forced to implement their own distributed termination detection. In this paper, we develop a simple set of ideas that allow work-stealing

to be efficiently extended to distributed memory. First, we introduce lifeline graphs: low-degree, low-diameter, fully connected directed graphs. Such graphs can be constructed from k-dimensional hypercubes. When a node is unable to find work after w unsuccessful steals, it quiesces after informing the outgoing edges in its lifeline graph. Quiescent nodes do not disturb other nodes. A quiesced node is reactivated when work arrives from a lifeline and itself shares this work with those of its incoming lifelines that are activated. Termination occurs precisely when computation at all nodes has quiesced. In a language such as X10, such passive distributed termination can be detected automatically using the finish construct – no application code is necessary. Our design is implemented in a few hundred lines of X10. On the binomial tree described in olivier:08, the program achieve 87% efficiency on an Infiniband cluster of 1024 Power7 cores, with a peak throughput of 2.37 GNodes/sec. It achieves 87% efficiency on a Blue Gene/P with 2048 processors, and a peak throughput of 0.966 GNodes/s. All numbers are relative to single core sequential performance. This implementation has been refactored into a reusable global load balancing framework. Applications can use this framework to obtain global load balance with minimal code changes. In summary, we claim: (a) the first formulation of UTS that does not involve application level global termination detection, (b) the introduction of lifeline graphs to reduce failed steals (c) the demonstration of simple lifeline graphs based on k-hypercubes, (d) performance with superior efficiency (or the same efficiency but over a wider range) than published results on UTS. In particular, our framework can deliver the same or better performance as an unrestricted random work-stealing implementation, while reducing the number of attempted steals.

[43] Dragos Sbîrlea, Zoran Budimlić, and Vivek Sarkar. "Bounded Memory Scheduling of Dynamic Task Graphs". In: *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*. PACT '14. Edmonton, AB, Canada: ACM, 2014, pp. 343–356. ISBN: 978-1-4503-2809-8. DOI: 10.1145/2628071.2628090. URL: http://doi.acm.org/10.1145/2628071.2628090.
It is now widely recognized that increased levels of parallelism is a necessary condition for improved application performance on multicore computers. However, as the number of cores increases, the memory-per-core ratio is expected to further decrease, making per-core memory efficiency of parallel programs an even more important concern in future systems. For many parallel applications, the memory requirements can be significantly larger than for their sequential counterparts and, more importantly, their memory utilization depends critically on the schedule used when running them. To address this problem we propose bounded memory scheduling (BMS) for parallel programs expressed as dynamic task graphs, in which an upper bound is imposed on the program's peak memory. Using the inspector/executor model, BMS tailors the set of allowable schedules to either guarantee that the program can be executed within the given memory bound, or throw an error during the inspector phase without running the computation if no feasible schedule can be found. Since solving BMS is NP-hard, we propose an approach in which we first use our heuristic algorithm, and if it fails we fall back on a more expensive optimal approach which is sped up by the best-effort result of the heuristic.

[44] Kirk Schloegel, George Karypis, and Vipin Kumar. "Multilevel Diffusion Schemes for Repartitioning of Adaptive Meshes". In: *Journal of Parallel and Distributed Computing* 47.2 (1997), pp. 109–124. ISSN: 0743-7315. DOI: https://doi.org/10.1006/jpdc.1997.1410. URL: http://www.sciencedirect.com/science/article/pii/S0743731597914106.
For a large class of irregular mesh applications, the structure of the mesh changes from one phase of the computation to the next. Eventually, as the mesh evolves, the adapted mesh has to be repartitioned to ensure good load balance. If this new graph is partitioned from scratch, it may lead to an excessive migration of data among processors. In this paper, we present schemes for computing repartitionings of adaptively refined meshes that perform diffusion of vertices in a multilevel framework. These schemes try to minimize vertex movement without significantly compromising the edge-cut. We present heuristics to control the tradeoff between edge-cut and vertex migration costs. We also show that multilevel diffusion produces results with improved edge-cuts over single-level diffusion, and is better able to make use of heuristics to control the tradeoff between edge-cut and vertex migration costs than single-level diffusion.

[45] Oksana Severiukhina et al. "Adaptive load balancing of distributed multi-agent simulations on heterogeneous computational infrastructures". In: *Procedia Computer Science* 119 (2017). 6th International Young Scientist Conference on Computational Science, YSC 2017, 01-03 November 2017, Kotka, Finland, pp. 139–146. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2017.11.170. URL: http://www.sciencedirect.com/science/article/pii/S1877050917323803.

Simulation of the agent-based model has several problems related to scalability, the accuracy of reproduction of motion. The increase in the number of agents leads to additional computations and hence the program run time also increases. This problem can be solved using distributed simulation and distributed computational environments such as clusters and supercomputers. The model objects must be divided into different processes and calculations to be able to be executed in parallel. This paper presents the research on an algorithm to balancing of computational load. The algorithm is based on a well-known genetic algorithm and performs optimization of matching between the model structure formed by multiple interconnected executable blocks of a distributed programming implementation of the model and network structure of the computational environment. Efficient placement of the model graphs nodes to heterogeneous computational ones leads to improvement in overall performance of the simulation and reducing of execution time.

[46] Bo Shao et al. "A Load Balancing Strategy for Monte Carlo Method in PageRank Problem". In: *Parallel Architecture, Algorithm and Programming*. Ed. by Guoliang Chen, Hong Shen, and Mingrui Chen. Singapore: Springer Singapore, 2017, pp. 594–609. ISBN: 978-981-10-6442-5.
PageRank algorithm is key component of a wide range of applications. Former study has demonstrated that PageRank problem can be effectively solved through Monte Carlo method. In this paper, we focus on efficiently parallel implementing Monte Carlo method for PageRank algorithm based on GPU. Aiming at GPU, a parallel implementation must consider instruction divergence on the single instruction multiple data (SIMD) compute units. Due to the fact that low-discrepancy sequences are determined sequences, we adopt the low-discrepancy sequences to simulate the random walks in PageRank computations in our load balancing strategy. Furthermore, we allocate each thread of a block to compute a random walk of each vertex with a same low-discrepancy sequence. As a result, no idle thread exists in the PageRank computations and warp execution efficiency is up to 99%. Moreover, our strategy loads the low-discrepancy sequences into shared memory to reduce the data fetch cost. The results of experiments show that our strategy can provide high efficiency for Monte Carlo method in PageRank problem in GPGPU environment.

[47] James D. Teresco, Karen D. Devine, and Joseph E. Flaherty. *2 Partitioning and Dynamic Load Balancing for the Numerical Solution of Partial Differential Equations*. 2006.
In parallel simulations, partitioning and load-balancing algorithms compute the distribution of application data and work to processors. The effectiveness of this distribution greatly influences the performance of a parallel simulation. Decompositions that balance processor loads while keeping the application communication costs low are preferred. Although a wide variety of partitioning and load-balancing algorithms have been developed, their effectiveness depends on the characteristics of the application using them. In this chapter, we review several partitioning algorithms, along with their strengths and weaknesses for various PDE applications. We also discuss current efforts toward improving partitioning algorithms for future applications and architectures.

[48] C. Torng, M. Wang, and C. Batten. "Asymmetry-Aware Work-Stealing Runtimes". In: *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. June 2016, pp. 40–52. DOI: 10.1109/ISCA.2016.14.
Amdahl's law provides architects a compelling reason to introduce system asymmetry to optimize for both serial and parallel regions of execution. Asymmetry in a multicore processor can arise statically (e.g., from core microarchitecture) or dynamically (e.g., applying dynamic voltage/frequency scaling). Work stealing is an increasingly popular approach to task distribution that elegantly balances task-based parallelism across multiple worker threads. In this paper, we propose asymmetry-aware work-stealing (AAWS) runtimes, which are carefully designed to exploit both the static and dynamic asymmetry in modern systems. AAWS runtimes use three key hardware/software techniques: work-pacing, work-sprinting, and work-mugging. Work-pacing and work-sprinting are novel techniques that combine a marginal-utility-based approach with integrated voltage regulators to improve performance and energy efficiency in high-and low-parallel regions. Work-mugging is a previously proposed technique that enables a waiting big core to preemptively migrate work from a busy little core. We propose a simple implementation of work-mugging based on lightweight user-level interrupts. We use a vertically integrated research methodology spanning software, architecture, and VLSI to make the case that holistically combining static asymmetry, dynamic asymmetry, and work-stealing runtimes can improve both performance and energy efficiency in future multicore systems.

[49] Cheng-Zhong Xu and Francis C. M. Lau. "Iterative Dynamic Load Balancing in Multicomputers". In: *The Journal of the Operational Research Society* 45.7 (1994), pp. 786–796. ISSN: 01605682, 14769360. URL: http://www.jstor.org/stable/2584287.

Dynamic load balancing in multicomputers can improve the utilization of processors and the efficiency of parallel computations through migrating the workload across processors at runtime. We present a survey and critique of dynamic load balancing strategies that are iterative: that is, workload migration is carried out through transferring processes across nearest neighbour processors. Iterative strategies have become prominent in recent years because of the increaasing popularity of point-to-point interconnection networks for multicomputers.

[50] C.Z. Xu and F.C.M. Lau. "The Generalized Dimension Exchange Method for Load Balancing in k-ary n-Cubes and Variants". In: *Journal of Parallel and Distributed Computing* 24.1 (1995), pp. 72–85. ISSN: 0743-7315. DOI: https://doi.org/10.1006/jpdc.1995.1007. URL: http://www.sciencedirect.com/science/article/pii/S0743731585710076.
The generalized dimension exchange (GDE) method is a fully distributed load balancing method that operates in a relaxation fashion for multicomputers with a direct communication network. It is parameterized by an exchange parameter lambda that governs the splitting of load between a pair of directly connected processors during load balancing. An optimal lambda would lead to the fastest convergence of the balancing process. Previous work has resulted in the optimal lambda for the binary n-cubes. In this paper, we derive the optimal lambda for the k-ary n-cube network and its variants-the ring, the torus, the chain, and the mesh. We establish the relationships between the optimal convergence rates of the method when applied to these structures, and conclude that the GDE method favors high-dimensional k-ary n-cubes. We also reveal the superiority of the GDE method to another relaxation-based method, the diffusion method. We further show through statistical stimulations that the optimal lambda do speed up the GDE balancing procedure significantly. Because of its simplicity, the method is readily implementable. We report on the implementation of the method in two data-parallel computations in which the improvement in performance due to GDE balancing is substantial.

[51] Jixiang Yang and Qingbi He. "Scheduling Parallel Computations by Work Stealing: A Survey". In: *International Journal of Parallel Programming* (Jan. 2017). ISSN: 1573-7640. DOI: 10.1007/s10766-016-0484-8. URL: https://doi.org/10.1007/s10766-016-0484-8.
Work stealing has been proven to be an efficient technique for scheduling parallel computations, and has been gaining popularity as the multiprocessor/multicore-processor load balancing technology of choice in both industry and academia. A review on the work stealing scheduling is provided from the perspective of scheduling algorithms, optimization of algorithm implementation and processor architecture oriented optimization. The future research trends and recommendations driven by theory, emerging applications and motifs, architecture and heterogeneous platforms are also provided.

[52] C. Zhang et al. "Dynamic load balancing on multi-GPUs system for big data processing". In: *2017 23rd International Conference on Automation and Computing (ICAC)*. Sept. 2017, pp. 1–6. DOI: 10.23919/IConAC.2017.8082085.
The powerful parallel computing capability of modern GPU (Graphics Processing Unit) processors has attracted increasing attentions of researchers and engineers who had conducted a large number of GPU-based acceleration research projects. However, current single GPU based solutions are still incapable of fulfilling the real-time computational requirements from the latest big data applications. Thus, the multi-GPU solution has become a trend for many real-time application attempts. In those cases, the computational load balancing over the multiple GPU nodes is often the key bottleneck that needs to be further studied to ensure the best possible performance. The existing load balancing approaches are mainly based on the assumption that all GPUs in the same system provide equal computational performance, and had fallen short to address the situations from heterogeneous multi-GPU systems. This paper presents a novel dynamic load balancing model for heterogeneous multi-GPU systems based on the fuzzy neural network (FNN) framework. The devised model has been implemented and demonstrated in a case study for improving the computational performance of a two dimensional (2D) discrete wavelet transform (DWT). Experiment results show that this dynamic load balancing model has enabled a high computational throughput that can satisfy the real-time and accuracy requirements from many big data processing applications.

[53] J. Zhang et al. "Dynamic Load Balancing Based on Constrained K-D Tree Decomposition for Parallel Particle Tracing". In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (Jan. 2018), pp. 954–963. ISSN: 1077-2626. DOI: 10.1109/TVCG.2017.2744059.
We propose a dynamically load-balanced algorithm for parallel particle tracing, which periodically attempts to evenly redistribute particles across processes based on k-d tree decomposition. Each process is assigned with (1) a statically partitioned, axis-aligned data block that partially overlaps with neighboring blocks in other