Multilevel Acyclic Partitioning of Directed Acyclic Graphs for Enhancing Data Locality

Julien Herrmann[1], Bora Uçar[2], Kamer Kaya[3], Aravind Sukumaran Rajam[4], Fabrice Rastello[5], P. Sadayappan[4], Ümit V. Çatalyürek[1]

[1]School of Computational Science and Engineering, Georgia Institute of Technology
[2]CNRS and LIP, ENS Lyon, France
[3]Computer Science and Engineering, Sabancı University, Turkey
[4]Department of Computer Science and Engineering, The Ohio State University
[5]INRIA Grenoble Rhone-Alpes, France

SIAM CSE

February 28th, 2017 – Atlanta, GA

Outline

1. Motivation
2. Acyclic Partitioning
3. Directed Multilevel Graph Partitioning
4. Experimental Results

Motivation

- Scheduling for task-based runtime systems by Ç et al.
- **Characterization of the Data Movement Complexity of Algorithms** by
  P. Sadayappan, A. Rountev, L-N. Pouchet, A. Sidiropoulos, N. Fauzia, V. Elango, and M. Ravishankar, The Ohio State University
  J. Ramanujam, Louisiana State University
  F. Rastello, INRIA-Grenoble

Motivation

- Data movement is much more expensive in computer systems than arithmetic operations (Flops)
  - Performance: latency as well as throughput
  - Energy

Motivation

- Data movement is much more expensive in computer systems than arithmetic operations (Flops)
  - Performance: latency as well as throughput
  - Energy
- Computational complexity alone (number of ops executed) cannot be sole (or even primary) criterion of algorithm choice

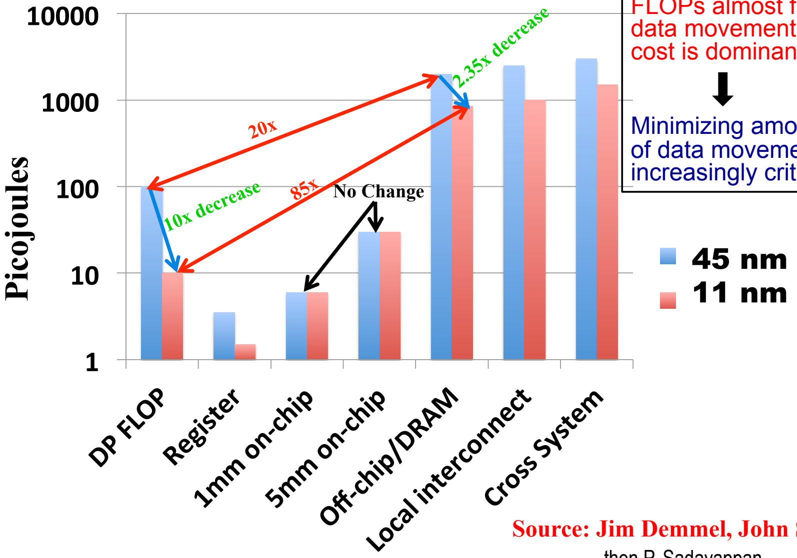
Motivation

- Data movement is much more expensive in computer systems than arithmetic operations (Flops)
  - Performance: latency as well as throughput
  - Energy
- Computational complexity alone (number of ops executed) cannot be sole (or even primary) criterion of algorithm choice
- But what is the inherent data movement complexity of an alg.?
  - Computational complexity well understood; invariant to transforms
  - Data access complexity is not well characterized today: cost is affected by code transformations and also capacity of registers/caches

Motivation

- Data movement is much more expensive in computer systems than arithmetic operations (Flops)
  - Performance: latency as well as throughput
  - Energy
- Computational complexity alone (number of ops executed) cannot be sole (or even primary) criterion of algorithm choice
- But what is the inherent data movement complexity of an alg.?
  - Computational complexity well understood; invariant to transforms
  - Data access complexity is not well characterized today: cost is affected by code transformations and also capacity of registers/caches
- Understanding data movement complexity is important:
  - Algorithm choice between alternatives e.g., will Krylov subspace solvers and FFTs continue to be as popular in the future?
  - Arch. parameters: minimum cache capacity and/or bus bw. needed to support inherent data movement needs of an alg.
  - Assessing manual/compiler optimizations: How much further improvement potential is there, beyond current optmizations?

Data Movement Cost: Energy Trends



FLOPs almost free; data movement cost is dominant

Minimizing amount of data movement increasingly critical

Source: Jim Demmel, John Shalf

then P. Sadayappan

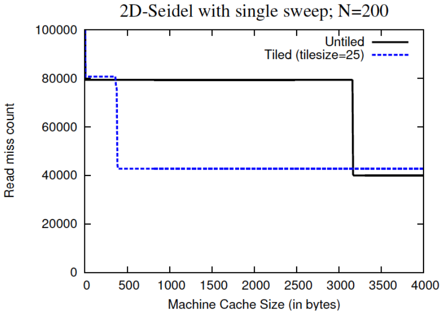
Computational vs Data Move Complexity

```
for (i=1; i<N-1; i++)
  for (j=1;j<N-1; j++)
    A[i][j] = A[i][j-1] + A[i-1][j];
```

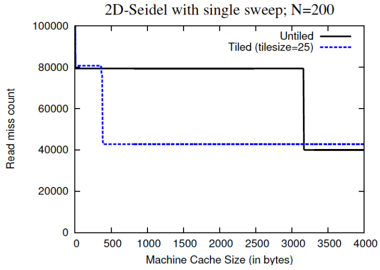
Untiled version

```
for(it = 1; it<N−1; it +=B)
 for(jt = 1; jt<N−1; jt +=B)
  for(i = it; i < min(it+B, N−1); i++)
   for(j = jt; j < min(jt+B, N−1); j++)
    A[i][j] = A[i−1][j] + A[i][j−1];
```

Tiled Version

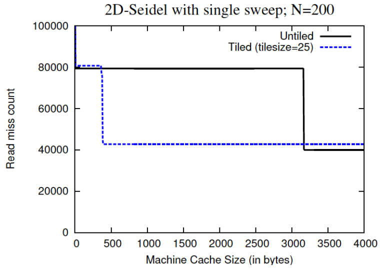


Computational vs Data Move Complexity



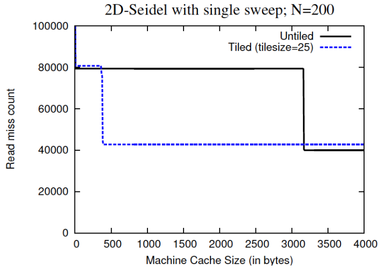
- Both have Comp. Complexity $(N-1)^2$ OPs.
  - Data movement cost different for two versions
  - Also depends on cache size

Computational vs Data Move Complexity



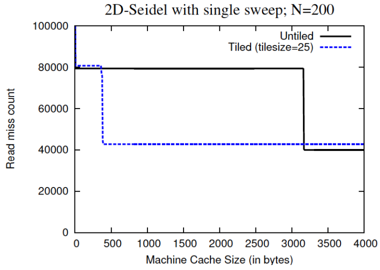
- Both have Comp. Complexity $(N-1)^2$ OPs.
  - Data movement cost different for two versions
  - Also depends on cache size
- Question: Can we achieve lower cache misses than this tiled version? How can we know when much further improvement is not possible?

Computational vs Data Move Complexity



- Both have Comp. Complexity $(N-1)^2$ OPs.
  - Data movement cost different for two versions
  - Also depends on cache size
- Question: Can we achieve lower cache misses than this tiled version? How can we know when much further improvement is not possible?
- Question: What is the lowest achievable data movement cost among all possible equivalent versions of a #computation?

Computational vs Data Move Complexity



- Both have Comp. Complexity $(N-1)^2$ OPs.
  - Data movement cost different for two versions
  - Also depends on cache size
- Question: Can we achieve lower cache misses than this tiled version? How can we know when much further improvement is not possible?
- Question: What is the lowest achievable data movement cost among all possible equivalent versions of a #computation?
- Current performance tools and methodologies do not address this

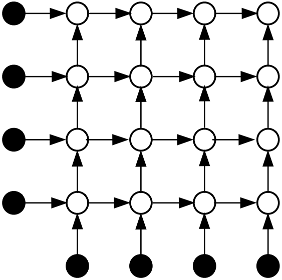
Modeling Data Move Complexity: CDAG

```
for (i=1; i<N-1; i++)
  for (j=1;j<N-1; j++)
    A[i][j] = A[i][j-1] + A[i-1][j];
```

Untiled version

```
for(it = 1; it<N−1; it +=B)
  for(jt = 1; jt<N−1; jt +=B)
    for(i = it; i < min(it+B, N−1); i++)
      for(j = jt; j < min(jt+B, N−1); j++)
        A[i][j] = A[i−1][j] + A[i][j−1];
```

Tiled Version



CDAG for N=6

- CDAG abstraction: Vertex = operation, edges = data dep.

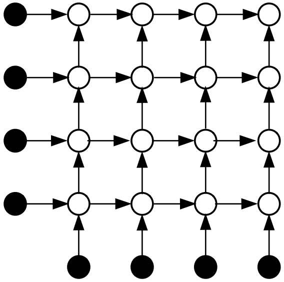
Modeling Data Move Complexity: CDAG

```
for (i=1; i<N-1; i++)
  for (j=1;j<N-1; j++)
    A[i][j] = A[i][j-1] + A[i-1][j];
```

Untiled version

```
for(it = 1; it<N−1; it +=B)
 for(jt = 1; jt<N−1; jt +=B)
  for(i = it; i < min(it+B, N−1); i++)
   for(j = jt; j < min(jt+B, N−1); j++)
    A[i][j] = A[i−1][j] + A[i][j−1];
```

Tiled Version



CDAG for N=6

- CDAG abstraction: Vertex = operation, edges = data dep.
- 2-level memory hierarchy with S fast mem locs. & infinite slow mem. locs.
  - To compute a vertex, predecessor must hold values in fast mem.
  - Limited fast memory $\Rightarrow$ computed values may need to be temporarily stored in slow memory and reloaded

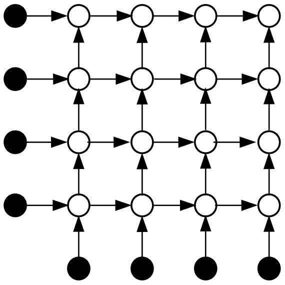
Modeling Data Move Complexity: CDAG

```
for (i=1; i<N-1; i++)
  for (j=1;j<N-1; j++)
    A[i][j] = A[i][j-1] + A[i-1][j];
```

Untiled version

```
for(it = 1; it<N−1; it +=B)
 for(jt = 1; jt<N−1; jt +=B)
  for(i = it; i < min(it+B, N−1); i++)
   for(j = jt; j < min(jt+B, N−1); j++)
    A[i][j] = A[i−1][j] + A[i][j−1];
```

Tiled Version



CDAG for N=6

- CDAG abstraction: Vertex = operation, edges = data dep.
- 2-level memory hierarchy with S fast mem locs. & infinite slow mem. locs.
  - To compute a vertex, predecessor must hold values in fast mem.
  - Limited fast memory ⇒ computed values may need to be temporarily stored in slow memory and reloaded
- Inherent data movement complexity of CDAG: Minimal #loads+#stores

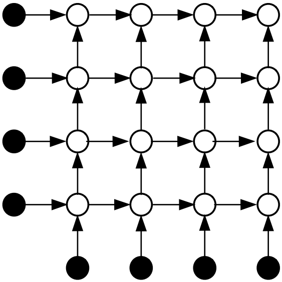
Modeling Data Move Complexity: CDAG

```
for (i=1; i<N-1; i++)
  for (j=1;j<N-1; j++)
    A[i][j] = A[i][j-1] + A[i-1][j];
```

Untiled version

```
for(it = 1; it<N−1; it +=B)
  for(jt = 1; jt<N−1; jt +=B)
    for(i = it; i < min(it+B, N−1); i++)
      for(j = jt; j < min(jt+B, N−1); j++)
        A[i][j] = A[i−1][j] + A[i][j−1];
```

Tiled Version



CDAG for N=6

Develop upper bounds on min-cost

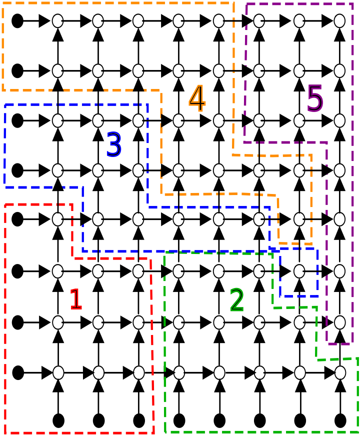
Minimum possible data movement cost?

No known effective solution to problem

Develop lower bounds on min-cost

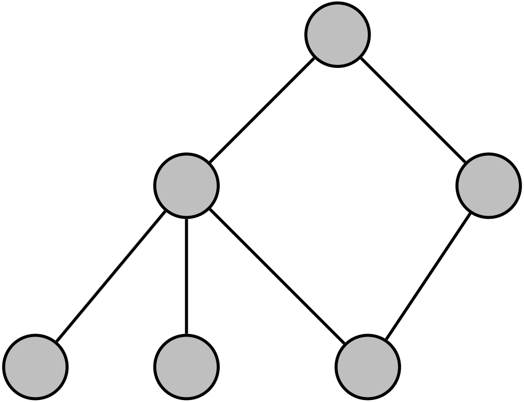
Data Movement Upper Bounds

- Perform acyclic partitioning of the CDAG
- Assign each node in a single acyclic part
- Acyclic partitioning of a CDAG $\approx$ Tiling the iteration space
- Each part is acyclic
  - Can be executed atomically
  - No cyclic data dependence among parts
- Topologically sorted order of the acyclic parts $\Rightarrow$ a valid execution order
- **To Do: Develop scalable distributed acyclic partitioning algorithm for CDAGs.**



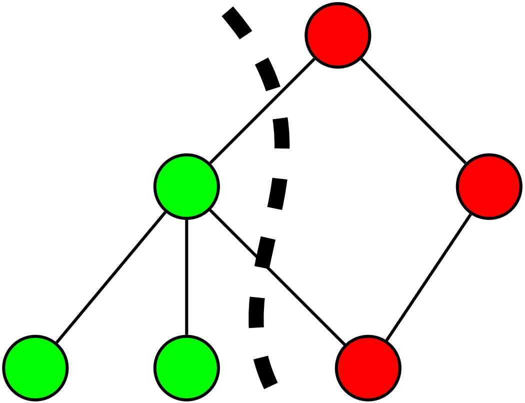
Outline

Balanced Acyclic Partitioning



Minimal edge cut:

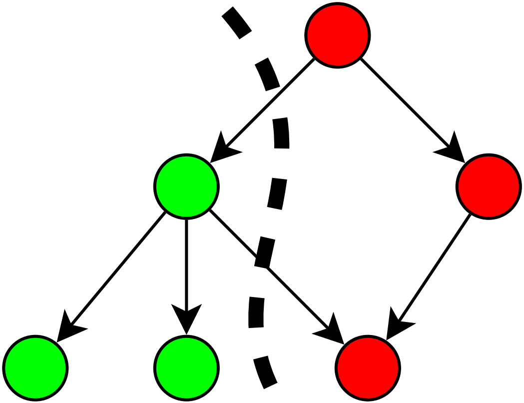
Balanced Acyclic Partitioning



Minimal edge cut:

- Undirected graph: 2

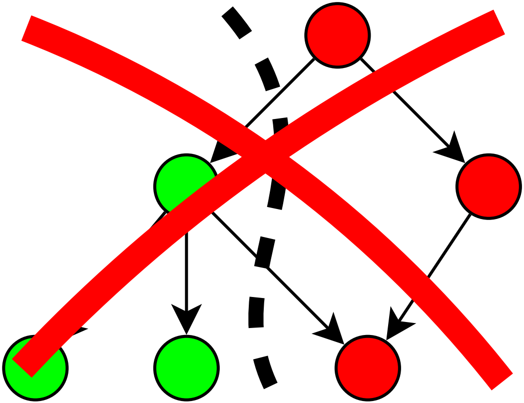
Balanced Acyclic Partitioning



Minimal edge cut:

- Undirected graph: 2

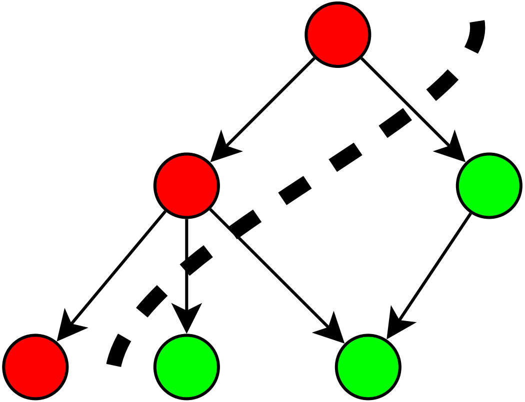
Balanced Acyclic Partitioning



Minimal edge cut:

- Undirected graph: 2

Balanced Acyclic Partitioning



Minimal edge cut:

- Undirected graph: 2
- Directed graph: 3

Objective Function

Objective 1

Minimize the edge cut between parts

Objective 2

Minimize the total volume of communication between parts (edge cut counting edges coming from the same node only once)

Objective 3

At the application level:

- Maximize the performance
- Minimize the cache miss count
- ...

Partition Constraint

Constraint 1

Upper bound on the weights of each part.

Constraint 2

Upper bound on the weight of each part plus the sum of weights of the boundary vertices that are sources of the part's incoming edges.

Constraint 3

There should exist a traversal of the graph such that **alive** data fit into the cache at any moment.

Fauzia et al. Algorithm

- Vertices are traversed in a topological order with tunable depth and breadth priorities.
- Vertices are assigned to the current partition set until the maximum number of vertices that would be active during the computation of the partition set reaches a specified cache size.

- Partition sizes can be larger than the size of the cache (Constraint 3).

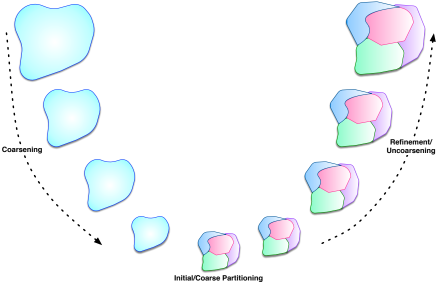
- This differs from our problem (Constraint 1).

Outline

Multilevel scheme

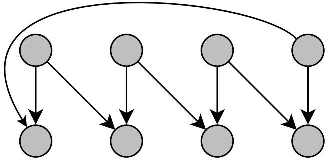
Three phases

- **Coarsening**: obtain smaller and similar graphs to the original, until either a minimum vertex count is reached or reduction on number of vertices is lower than a threshold.
- **Initial Partitioning**: find a solution for the smallest graph.
- **Uncoarsening**: Project the initial solution to the coarser graphs and refine it iteratively until a solution for the original graph obtained.



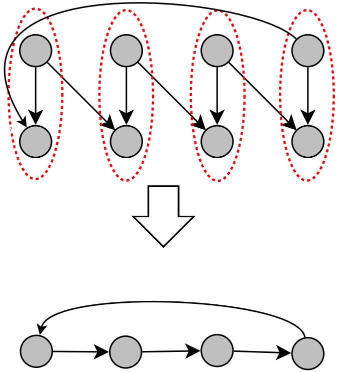
Coarsening

- Make sure not to create any cycle when matching



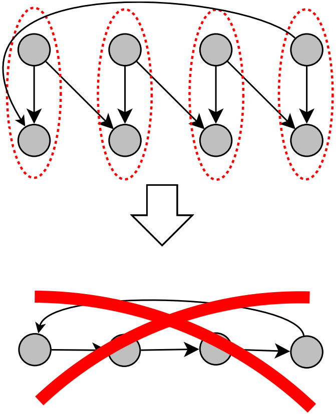
Coarsening

- Make sure not to create any cycle when matching



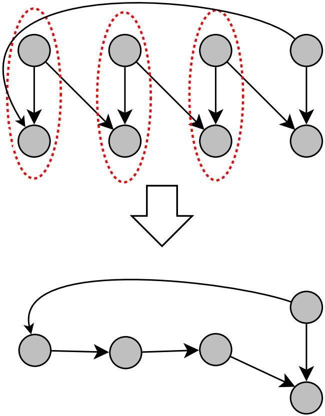
Coarsening

- Make sure not to create any cycle when matching



Coarsening

- Make sure not to create any cycle when matching
- Find optimal matching ⇒ too costly.



Coarsening

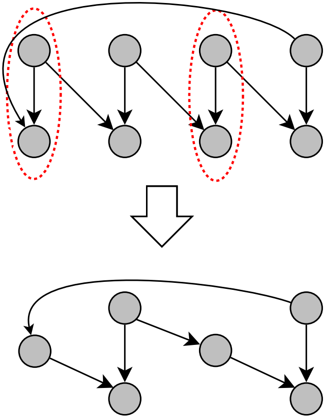
- Make sure not to create any cycle when matching
- Find optimal matching $\Rightarrow$ too costly.

Matching Restriction

Let $G = (V, E)$ be a CDAG and $M = \{(u_1, v_1), ..., (u_k, v_k)\}$ a matching such that:

- $(u_i, v_j) \in M$, $\text{top\_level}(v_i) = \text{top\_level}(u_i) + 1$
- any pair of (u_i, v_i) and $(u_j, v_j) \in M$, either
  - (u_i, v_j) not in E or
  - $\text{top\_level}(u_i) \neq \text{top\_level}(v_j) + 1$

Then, the coarse graph is acyclic.



Initial partitioning

Kernighan's Algorithm (1971)

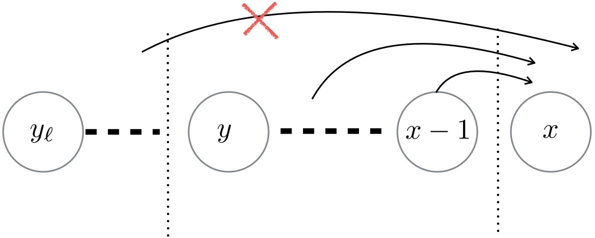
- Given a total order of vertices, vertex weights, edge costs, an upper bound on part weights,
- ..finds a cut so that part weights respect the upper bound and minimizes the edge cut, where the parts are contiguous.

We find a topological order with an attempt to reduce the maximum edge cut at a point (heuristically).

Then, feed this to Kernighan's algorithm.

Kernighan's Algorithm: Dynamic Programing

$$T(x) = \min_y \{T(y) + C(x, y)\}$$



$T(x)$: the best cost of cutting right before x.

$C(x, y)$: the additional cut edges at x, given the previous cut was at y. Do not count twice.

y_ℓ: the weight of the part $y_\ell, \ldots, x-1$ is acceptable, but $y_{\ell-1}, \ldots, x-1$ is not.

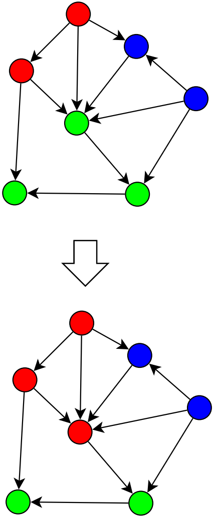
Uncoarsening

- Moving a node to another partition set can violate acyclicity

Refinement Restriction

- Define a topological order among parts.
- A node can only be moved to the part of its incoming nodes with the highest rank in the topological order or the part of its outgoing nodes with the smallest rank in the topological order.

Then, the refinement does not violate acyclicity.



- Nodes are moved as long as balance constraints are matched and edge cut is improving.

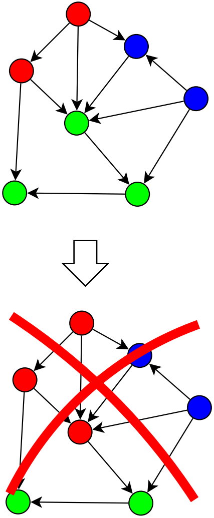
Uncoarsening

- Moving a node to another partition set can violate acyclicity

Refinement Restriction

- Define a topological order among parts.
- A node can only be moved to the part of its incoming nodes with the highest rank in the topological order or the part of its outgoing nodes with the smallest rank in the topological order.

Then, the refinement does not violate acyclicity.



- Nodes are moved as long as balance constraints are matched and edge cut is improving.

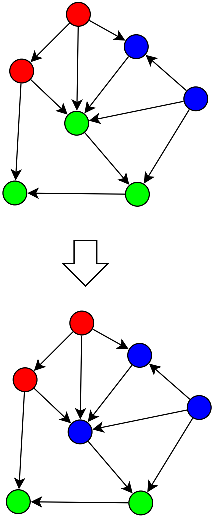
Uncoarsening

- Moving a node to another partition set can violate acyclicity

Refinement Restriction

- Define a topological order among parts.
- A node can only be moved to the part of its incoming nodes with the highest rank in the topological order or the part of its outgoing nodes with the smallest rank in the topological order.

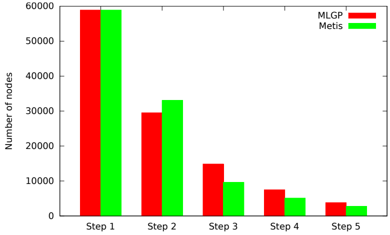
Then, the refinement does not violate acyclicity.

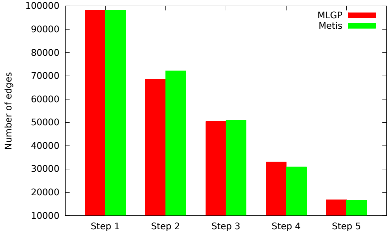


- Nodes are moved as long as balance constraints are matched and edge cut is improving.

Outline

Coarsening quality: Graph Jacobi-1d





- Coarsening ratios of Metis and dMLGP are very similar.
- Directed coarsening does not seem to be too restrictive.

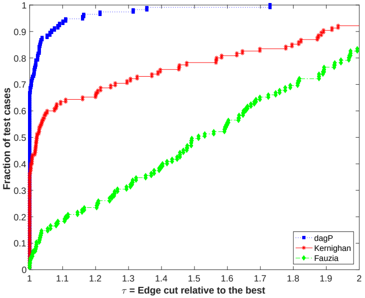
Graphs Properties

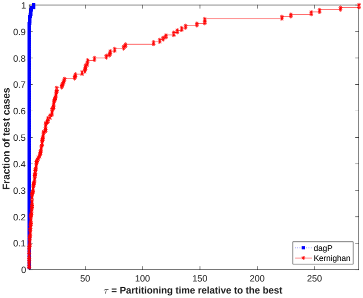
Instances from the Polyhedral Benchmark Suite.

Graph	Parameters	#*vertex*	#*edge*	out-deg.	deg.
`2mm`	P=10, Q=20, R=30, S=40	36,500	62,200	40	1.704
`3mm`	P=10, Q=20, R=30, S=40, T=50	111,900	214,600	40	1.918
`adi`	T=20, N=30	596,695	1,059,590	109,760	1.776
`atax`	M=210, N=230	241,730	385,960	230	1.597
`covariance`	M=50, N=70	191,600	368,775	70	1.925
`doitgen`	P=10, Q=15, R=20	123,400	237,000	150	1.921
`durbin`	N=250	126,246	250,993	252	1.988
`fdtd-2d`	T=20, X=30, Y=40	256,479	436,580	60	1.702
`gemm`	P=60, Q=70, R=80	1,026,800	1,684,200	70	1.640
`gemver`	N=120	159,480	259,440	120	1.627
`gesummv`	N=250	376,000	500,500	500	1.331
`heat-3d`	T=40, N=20	308,480	491,520	20	1.593
`jacobi-1d`	T=100, N=400	239,202	398,000	100	1.664
`jacobi-2d`	T=20, N=30	157,808	282,240	20	1.789
`lu`	N=80	344,520	676,240	79	1.963
`ludcmp`	N=80	357,320	701,680	80	1.964
`mvt`	N=200	200,800	320,000	200	1.594
`seidel-2d`	M=20, N=40	261,520	490,960	60	1.877
`symm`	M=40, N=60	254,020	440,400	120	1.734
`syr2k`	M=20, N=30	111,000	180,900	60	1.630
`syrk`	M=60, N=80	594,480	975,240	81	1.640
`trisolv`	N=400	240,600	320,000	399	1.330
`trmm`	M=60, N=80	294,570	571,200	80	1.939

Experimental Results

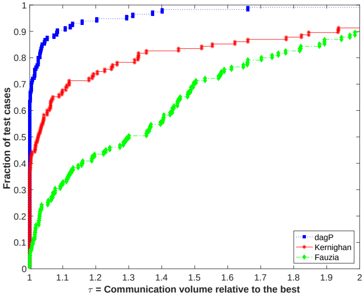
- Average results on 100 runs.
- Imbalance ratio of 3%.

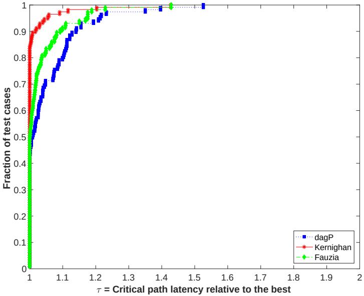




Results

- Inter-partitions edges have a weight of 11 nanoseconds to model L3 cache latency.
- Intra-partitions edges have a weight of 1 nanoseconds to model L1 cache latency.
- Vertices have a latency of 1 nanoseconds to model task execution.





Summary and Ongoing/Future Work

Data Movement

- Data movement costs will be increasingly dominant over computation costs, for both performance and energy/power
  - Important to understand inherent constraints on minimal possible data movement for an algorithm as a function of storage capacity
- Need advances in theory and software tools for modeling data movement complexity, and methodologies for application to algorithm analysis and algorithm-architecture co-design
  - Significant benefit of lower bounds analysis: schedule-independent, unlike standard performance modeling; especially powerful for analysis of composite applications

Summary and Ongoing/Future Work

Data Movement

- Data movement costs will be increasingly dominant over computation costs, for both performance and energy/power
  - Important to understand inherent constraints on minimal possible data movement for an algorithm as a function of storage capacity
- Need advances in theory and software tools for modeling data movement complexity, and methodologies for application to algorithm analysis and algorithm-architecture co-design
  - Significant benefit of lower bounds analysis: schedule-independent, unlike standard performance modeling; especially powerful for analysis of composite applications

Directed Graph Partitioning

- Implement agglomerative matching, i.e., clustering.
- Use directed graph partitioning to automaticly improve data locality for compiler optimizations.

Thanks

Thanks

To P. Sadayappan for sharing his motivation slides.

More information

contact : umit@gatech.edu

visit: `http://cc.gatech.edu/~umit`