LS-GPart: a global, distributed ordering library

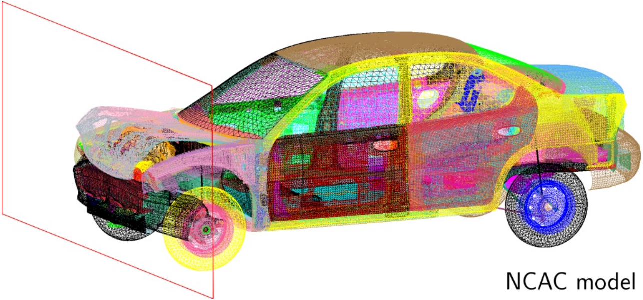
Cleve Ashcraft, François-Henry Rouet

Livermore Software Technology Corporation

Sparse Days 2017, September 7th, 2017

LS-DYNA

- Originated at the Lawrence Livermore Lab (DYNA3D, 1976).
- Applications: automotive crash and occupant safety, metal forming, CFD, FSI, electromagnetism, acoustics, thermal…
- Finite elements, boundary elements, meshless, SPH…
- About 50% of the explicit crash market, a share of the growing implicit market.
- Linear algebra team: Bob Lucas, Cleve Ashcraft, Roger Grimes, Clément Weisbecker, F.-H. Rouet, Eugene Vecharynski.



NCAC model

Fill-reducing ordering

Implicit solver in LS-Dyna:

- The vast majority of our matrices are symmetric.
- We tried early versions of ParMETIS and PT-Scotch but observed quality degradation (parallel vs serial).
- By default: serial METIS (1 trial per compute node), sometimes MMD. For very large problems, METIS runs out of memory, and MMD doesn't compete with Nested Dissection.

Goals:

- Memory scalability.
- Good quality regardless of number of processors.
- Reorder separators/fronts for Block Low-Rank factorization.

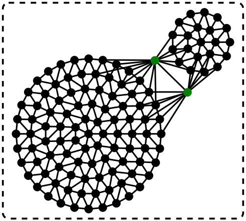
LS-GPart

- Nested dissection / recursive bisection.
- Not multilevel. A “global” partitioner.
- Distributed implementation, no global data, no $O(N)$ vector.
- Typical input: weighted compressed graph. Vertex of the compressed graph $\equiv$ 3, 6,… rows/cols of the matrix (degrees of freedom).

Algorithm

Until no subgraph is shared by multiple processors, or too large:

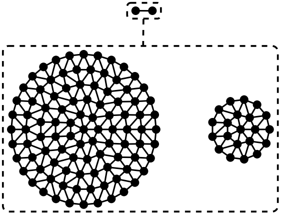
- Any (nearly) dense rows?



Algorithm

Until no subgraph is shared by multiple processors, or too large:

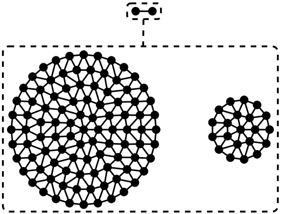
- Any (nearly) dense rows?
  Remove them and put in parent graph.



Algorithm

Until no subgraph is shared by multiple processors, or too large:

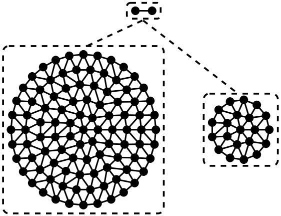
- Any (nearly) dense rows?
  Remove them and put in parent graph.
- Is the graph disconnected?



Algorithm

Until no subgraph is shared by multiple processors, or too large:

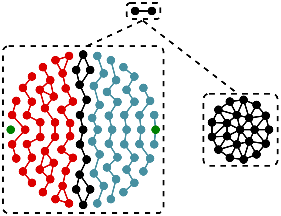
- Any (nearly) dense rows?
  Remove them and put in parent graph.
- Is the graph disconnected?
  Create graph objects for every component, redistribute to different processes.



Algorithm

Until no subgraph is shared by multiple processors, or too large:

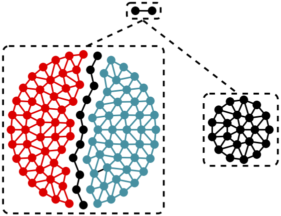
- Any (nearly) dense rows?
  Remove them and put in parent graph.
- Is the graph disconnected?
  Create graph objects for every component, redistribute to different processes.
- Find candidate separators
  Based on half-level sets.



Algorithm

Until no subgraph is shared by multiple processors, or too large:

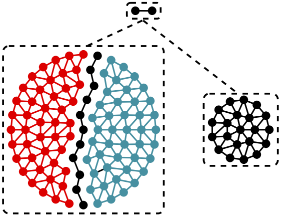
- Any (nearly) dense rows?
  Remove them and put in parent graph.
- Is the graph disconnected?
  Create graph objects for every component, redistribute to different processes.
- Find candidate separators
  Based on half-level sets.
- Improve candidate separators
  With maxflow or block-trimming.



Algorithm

Until no subgraph is shared by multiple processors, or too large:

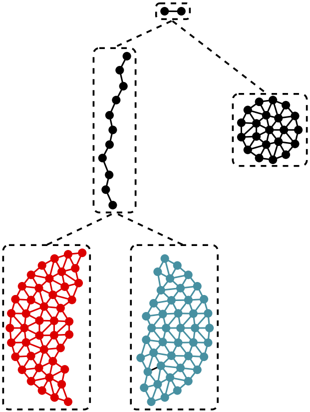
- Any (nearly) dense rows?
  Remove them and put in parent graph.
- Is the graph disconnected?
  Create graph objects for every component, redistribute to different processes.
- Find candidate separators
  Based on half-level sets.
- Improve candidate separators
  With maxflow or block-trimming.
- Pick best partition.



Algorithm

Until no subgraph is shared by multiple processors, or too large:

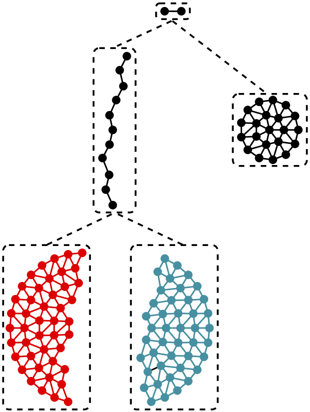
- Any (nearly) dense rows?
  Remove them and put in parent graph.
- Is the graph disconnected?
  Create graph objects for every component, redistribute to different processes.
- Find candidate separators
  Based on half-level sets.
- Improve candidate separators
  With maxflow or block-trimming.
- Pick best partition.
- Split, redistribute to disjoint sets of processors.



Algorithm

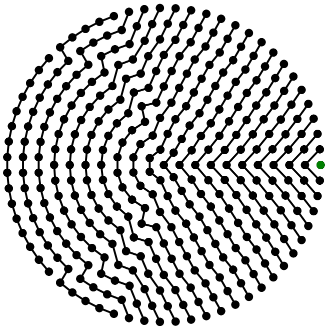
Until no subgraph is shared by multiple processors, or too large:

- Any (nearly) dense rows?
  Remove them and put in parent graph.
- Is the graph disconnected?
  Create graph objects for every component, redistribute to different processes.
- Find candidate separators
  Based on half-level sets.
- Improve candidate separators
  With maxflow or block-trimming.
- Pick best partition.
- Split, redistribute to disjoint sets of processors.
- Recurse on subgraphs owned by multiple processors.



GENeralized Automatic Nested Dissection [George & Liu '81]:

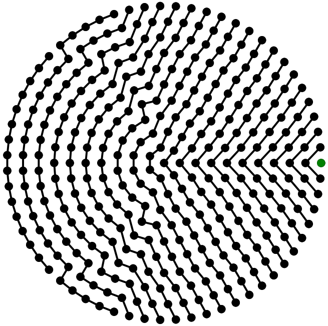
- Find a pseudo-peripheral node s ("source").



Finding an initial separator – 1

GENeralized Automatic Nested Dissection [George & Liu '81]:

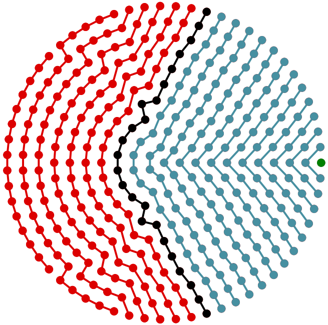
- Find a pseudo-peripheral node s ("source").
- Run BFS from s. level(u)=dist(u,s).
  Level set L_i: nodes with the same level i.
  Cuthill-McKee: reordering along $\{L_0, L_1, \ldots, L_{\text{diam}}\}$ makes the matrix block-tridiagonal.
  Every level set defines a separator S (often minimal); $B = \{L_0, \ldots, L_{i-1}\}$ and $W = \{L_{i+1}, \ldots, L_{\text{diam}}\}$ are candidate subdomains.



Finding an initial separator – 1

GENeralized Automatic Nested Dissection [George & Liu '81]:

- Find a pseudo-peripheral node s ("source").
- Run BFS from s. level(u)=dist(u,s).
  Level set L_i: nodes with the same level i.
  Cuthill-McKee: reordering along $\{L_0, L_1, \ldots, L_{\text{diam}}\}$ makes the matrix block-tridiagonal.
  Every level set defines a separator S (often minimal); $B = \{L_0, \ldots, L_{i-1}\}$ and $W = \{L_{i+1}, \ldots, L_{\text{diam}}\}$ are candidate subdomains.
- Pick the level set that minimizes a cost function, e.g., cost$(B, S, W) =$

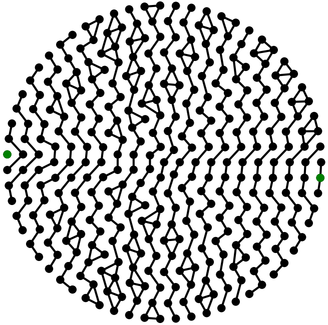


$$\begin{cases} +\infty & \textbf{if } \frac{\max(|B|,|W|)}{\min(|B|,|W|)} > \alpha \\ |S| \left(1 + \beta \frac{||B|-|W||}{|B|+|S|+|W|}\right) & \textbf{otherwise} \end{cases}$$

Finding an initial separator – 2

Our approach:

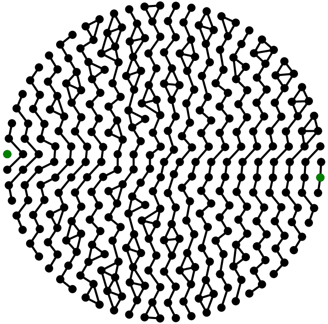
- We use two sources s and t and define half-level(u)=dist(u,s)-dist(u,t). Implicitly, reordering following half-level sets makes the matrix pentadiagonal. Two consecutive half-level sets define a separator. Not always minimal!



Finding an initial separator – 2

Our approach:

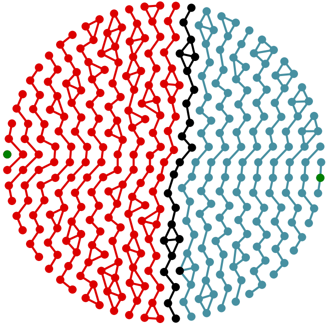
- We use two sources s and t and define half-level(u)=dist(u,s)-dist(u,t). Implicitly, reordering following half-level sets makes the matrix pentadiagonal. Two consecutive half-level sets define a separator. Not always minimal!
- Empirically, half-level sets yield better separators than single-level sets.



Finding an initial separator – 2

Our approach:

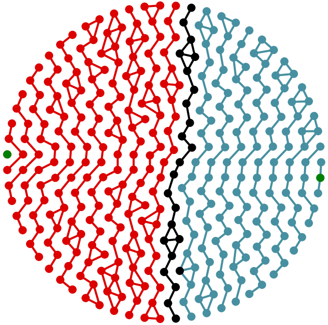
- We use two sources s and t and define half-level(u)=dist(u,s)-dist(u,t). Implicitly, reordering following half-level sets makes the matrix pentadiagonal. Two consecutive half-level sets define a separator. Not always minimal!
- Empirically, half-level sets yield better separators than single-level sets.
- For a given pair of sources, we find a separator using the same cost function.



Finding an initial separator – 2

Our approach:

- We use two sources s and t and define half-level(u)=dist(u,s)-dist(u,t). Implicitly, reordering following half-level sets makes the matrix pentadiagonal. Two consecutive half-level sets define a separator. Not always minimal!
- Empirically, half-level sets yield better separators than single-level sets.
- For a given pair of sources, we find a separator using the same cost function.
- We start with multiple sources, typically $O(10)$. We perform the multiple BFS in one shot to hide latency (number of communication steps = graph diameter). The number of pairs of sources and candidate partitions is quadratic in the number of sources.

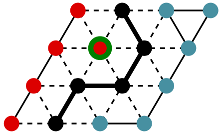


Improving a candidate separator – 1

A separator can be “straightened out” using selective expansion.

Add vertices to the separator based on their ratio $|\partial u \cap S| / |\partial u|$: number of neighbors in the separator vs total number of neighbors (“cutting corners”).

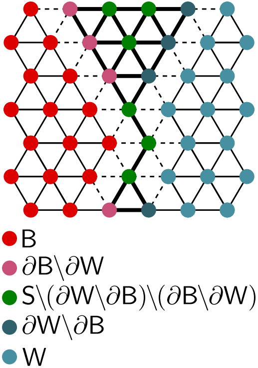
Multiple passes can be used. We set a limit on the size of the new wide separator relative to the original one.



Improving a candidate separator – 2

A non-minimal separator can improved/contracted using:

- Block-trimming: try putting $\partial B \setminus \partial W$ in B instead of S. Same with W.
  Cheap: 1 communication step per pass.



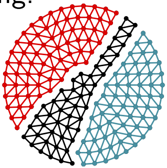
Improving a candidate separator – 2

A non-minimal separator can improved/contracted using:

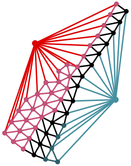
- Block-trimming: try putting $\partial B \setminus \partial W$ in B instead of S. Same with W.
  Cheap: 1 communication step per pass.
- Maxflow [Ashcraft & Liu '98]

Graph:	Network:
B, W	source s, sink t
Vertex u in S	$u^- \xrightarrow{\lvert u\rvert} u^+$
Edge from B to S	$s \xrightarrow{+\infty} u^-$
Edge from S to W	$u^+ \xrightarrow{+\infty} t$
Edge (u, v) in S	$u^- \xrightarrow{\lvert u\rvert} u^+$, $v^- \xrightarrow{\lvert v\rvert} v^+$, $u^+ \xrightarrow{+\infty} v^-$, $v^+ \xrightarrow{+\infty} u^-$

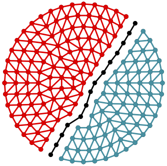
Much more expensive than block-trimming.



Initial separator.



Network & mincut.

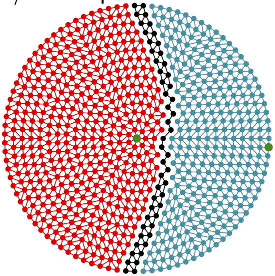


Improved separator.

Improving a candidate separator – 3

We perform cycles of expansions-contractions:

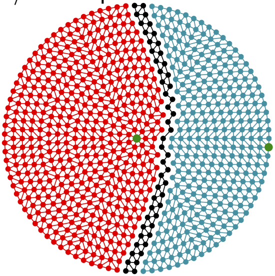
1/ Initial partition:



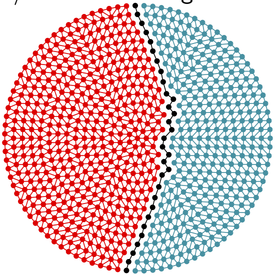
Improving a candidate separator – 3

We perform cycles of expansions-contractions:

1/ Initial partition:



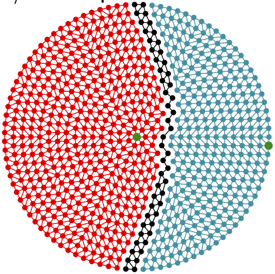
2/ Block-trimming:



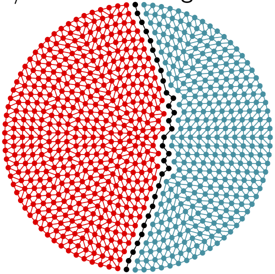
Improving a candidate separator – 3

We perform cycles of expansions-contractions:

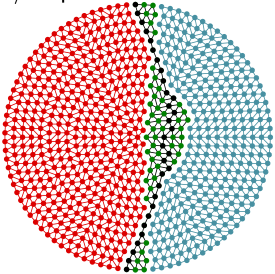
1/ Initial partition:



2/ Block-trimming:



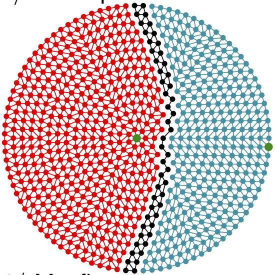
3/ Expansion:



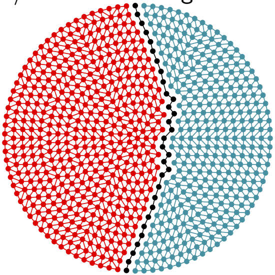
Improving a candidate separator – 3

We perform cycles of expansions-contractions:

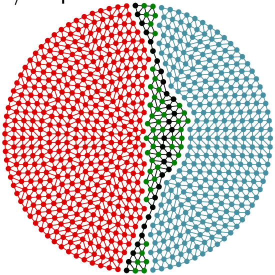
1/ Initial partition:



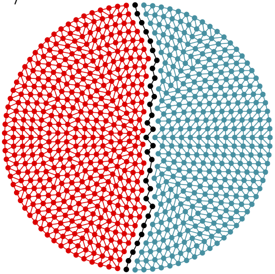
2/ Block-trimming:



3/ Expansion:



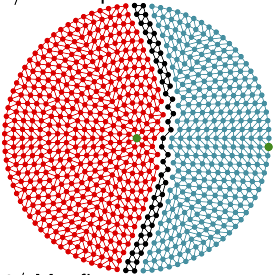
4/ Maxflow:



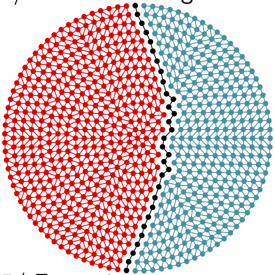
Improving a candidate separator – 3

We perform cycles of expansions-contractions:

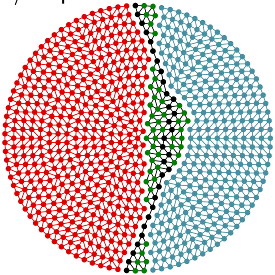
1/ Initial partition:



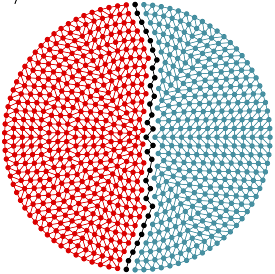
2/ Block-trimming:



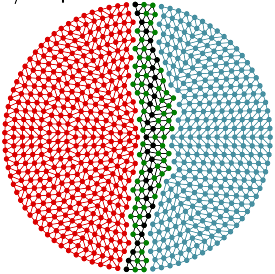
3/ Expansion:



4/ Maxflow:

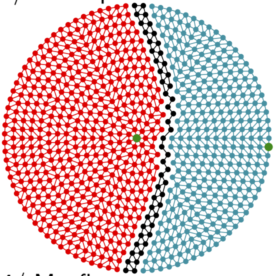


5/ Expansion:

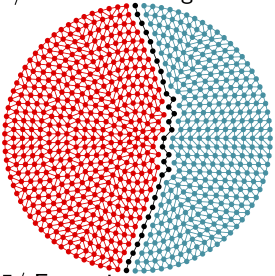


We perform cycles of expansions-contractions:

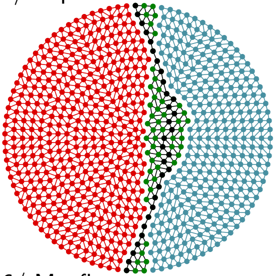
1/ Initial partition:



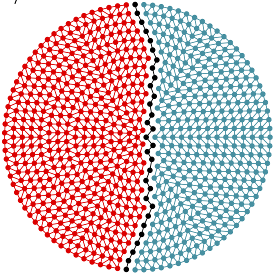
2/ Block-trimming:



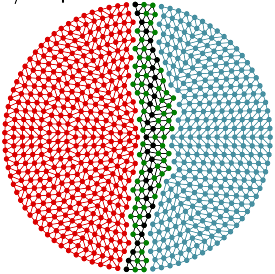
3/ Expansion:



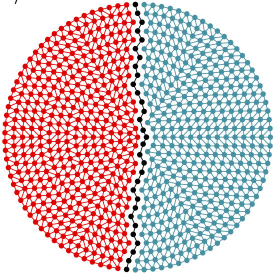
4/ Maxflow:



5/ Expansion:



6/ Maxflow:



Implementation

Data structures:

- Graph is fully distributed. A process knows only about its vertices I_q, ∂I_q (neighbors), and the edges in $I_q \times I_q$ and $\partial I_q \times I_q$. Ownership map is also distributed; no $O(N)$ vectors.
- Distance matrix, candidate partitions, etc. also distributed.

Implementation

Data structures:

- Graph is fully distributed. A process knows only about its vertices I_q, ∂I_q (neighbors), and the edges in $I_q \times I_q$ and $\partial I_q \times I_q$. Ownership map is also distributed; no $O(N)$ vectors.
- Distance matrix, candidate partitions, etc. also distributed.

Implementation: BSP style, `MPI_Alltoallv` is our workhorse.

Implementation

Data structures:

- Graph is fully distributed. A process knows only about its vertices I_q, ∂I_q (neighbors), and the edges in $I_q \times I_q$ and $\partial I_q \times I_q$. Ownership map is also distributed; no $O(N)$ vectors.
- Distance matrix, candidate partitions, etc. also distributed.

Implementation: BSP style, `MPI_Alltoallv` is our workhorse.

Initial separators construction:

- We use both single and dual sources.
- Pseudo-peripheral nodes: random sources, BFS to build distance information, discard sources too close from each other, replace them with new sources based on distance information.

Implementation

Data structures:

- Graph is fully distributed. A process knows only about its vertices I_q, ∂I_q (neighbors), and the edges in $I_q \times I_q$ and $\partial I_q \times I_q$. Ownership map is also distributed; no $O(N)$ vectors.
- Distance matrix, candidate partitions, etc. also distributed.

Implementation: BSP style, `MPI_Alltoallv` is our workhorse.

Initial separators construction:

- We use both single and dual sources.
- Pseudo-peripheral nodes: random sources, BFS to build distance information, discard sources too close from each other, replace them with new sources based on distance information.

Improving separators: maxflow and block-trimming (latter preferred).

Implementation

Data structures:

- Graph is fully distributed. A process knows only about its vertices I_q, ∂I_q (neighbors), and the edges in $I_q \times I_q$ and $\partial I_q \times I_q$. Ownership map is also distributed; no $O(N)$ vectors.
- Distance matrix, candidate partitions, etc. also distributed.

Implementation: BSP style, `MPI_Alltoallv` is our workhorse.

Initial separators construction:

- We use both single and dual sources.
- Pseudo-peripheral nodes: random sources, BFS to build distance information, discard sources too close from each other, replace them with new sources based on distance information.

Improving separators: maxflow and block-trimming (latter preferred).

Recursive bisection:

- We hand over the leaves of the tree to METIS.
- Top-level separators are the same regardless of #procs.

Test problems

Problem	Order	Entries	Application
Hex	12.6k	81.1k	Spot weld failure analysis
Dubcova1$^{(*)}$	16.1k	134.6k	High-order discretization
bmw7st_1$^{(*)}$	141.3k	3.7M	Car body static analysis
ptwk$^{(*)}$	217.9k	5.9M	Pressurized wind tunnel
Cylinders	506.8k	6.9M	Nested cylinders (solids)
audikw_1$^{(*)}$	943.7k	39.3M	Crankshaft model
Pickup	1.0M	6.7M	Pickup truck (shells, solids...)
Transport$^{(*)}$	1.6M	23.5M	Coupled flow and transport
3D grid	1.7M	17.1M	120^3 grid, 19-pt stencil
Machine	3.4M	48.6M	Cardboard bending machine (solids)
Impeller	7.0M	94.8M	24 fan blade impeller (solids)
Engine	11.1M	141.2M	Whole jet engine (solids)

$^{(*)}$: U.Fl. Collection. Others: LSTC applications; compressed graphs.

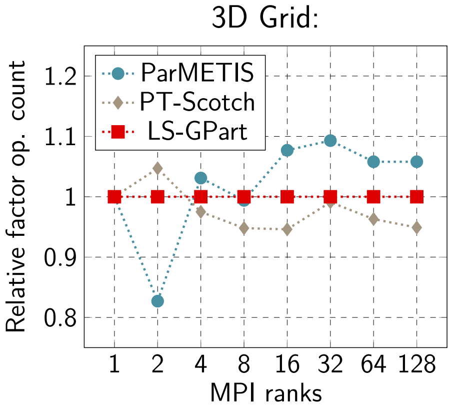
Ordering quality

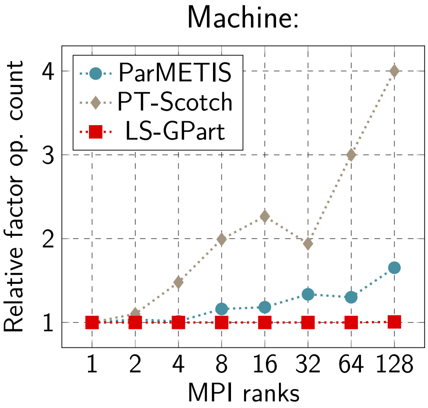
PT-Scotch 6.0.4, ParMETIS 4.0.3, METIS 5.1.0. 8 MPI ranks:

	Factor size w.r.t. METIS			Op. count w.r.t. METIS		
Problem	ParMETIS	PT-Scotch	LS-GPart	ParMETIS	PT-Scotch	LS-GPart
Hex	**0.99**	1.90	1.05	**0.96**	7.00	1.15
Dubcova1	1.02	1.27	**1.01**	1.07	1.61	**1.01**
bmw7st_1	**1.07**	1.15	1.08	1.43	1.91	**1.33**
ptwk	1.03	1.05	**1.03**	1.13	1.20	**1.13**
Cylinders	1.07	1.28	**0.99**	1.39	1.79	**0.99**
audikw_1	1.07	1.06	**1.04**	1.22	1.22	**1.09**
Pickup	**1.01**	1.25	1.13	**1.26**	2.05	1.95
Transport	**1.03**	1.08	1.03	1.14	1.19	**1.06**
3D grid	1.05	1.21	**0.99**	1.17	1.53	**0.98**
Machine	**1.00**	1.07	1.06	**1.05**	1.33	1.20
Impeller	**1.01**	1.24	1.04	**1.04**	1.87	1.06
Engine	**1.02**	1.26	1.08	**1.15**	2.33	1.42

Quality vs number of processes

Factor operation count for two problems:



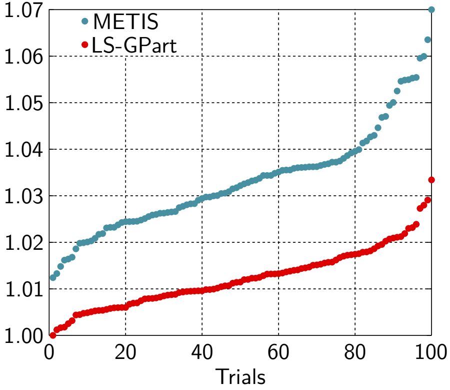


Our separators don't depend on number of MPI ranks nor input distribution (only vertex labeling, used to pick random sources).

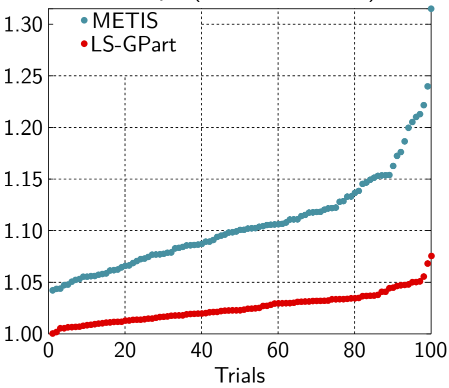
Influence of the random seed

LS-GPart (3 levels/8 domains) vs METIS, Dubcova problem:

Factor size (relative to best).



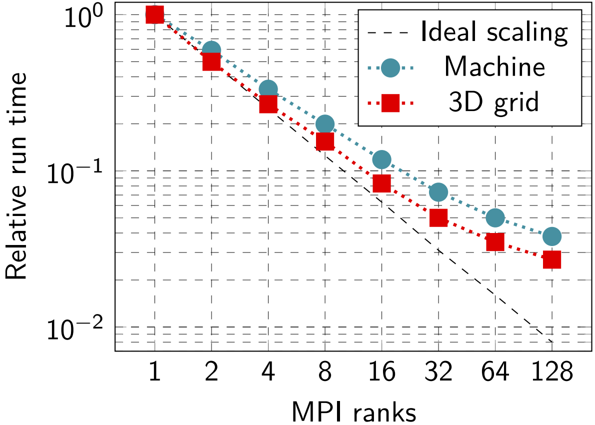
Factor ops (relative to best).



- METIS: wide variability. We often advise users to use 4 trials.
- LS-GPart more consistent.

Strong scaling

Parallel performance for two problems:



Remarks:

- Bottlenecks: BFS (1D parallelization), separator expansion.
- Slower than serial METIS for small numbers of processors; typically we catch up for 16 MPI ranks.

Conclusion

LS-GPart:

- Non-multilevel parallel nested dissection based on half-level sets.
- Preliminary experiments show reasonable ordering quality and parallel scalability.
- Separators are built with smoothness in mind. Useful for low-rank factorizations?

Work in progress:

- Approximate BFS to reduce cost.
- Raceahead separator expansion algorithm.
- Parallel symbolic factorization in the same code.
- Edge-based multisection using the same framework for separators/frontal matrices (for low-rank factorizations).

End

Thank you for your attention!

Any questions?