UMPa: A Multi-objective, multi-level partitioner for communication minimization

Ümit V. Çatalyürek[1], Mehmet Deveci[1], Kamer Kaya[1], Bora Uçar[2]

[1]Department of Biomedical Informatics, The Ohio State University
[2]LIP, ENS Lyon

Introduction

- Problem: distributing communicating tasks, modeled as a graph, among processing units.
  - Balanced load distribution
  - Good communication pattern
- Objective functions from the literature:
  - Total communication volume
  - Maximum communication volume
  - Maximum send volume

Problem: Input and objective

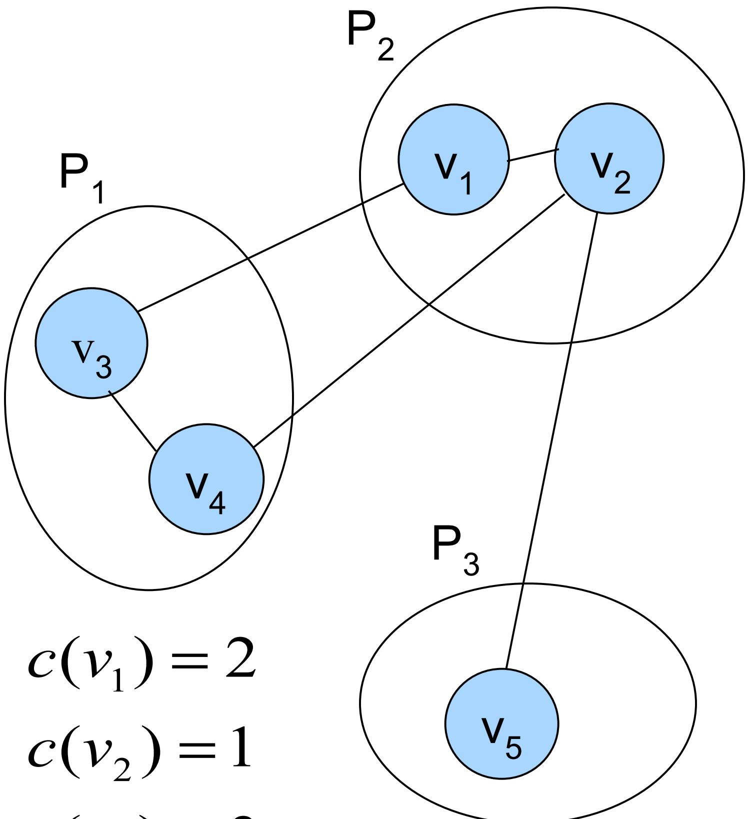
- Input: task graph G=(V,E)
  - V: vertex set representing a set of tasks
  - E: edge set representing task communications
- Objective: Find a partition $\prod = \{P_1, P_2, \ldots, P_K\}$ of the tasks s.t.

$$\max_k \left(\sum_{v \in P_k} c(v) \times f(v) \right)$$

is minimized

  - c(v): volume of each transfer sent by v.
  - f(v): number of parts that requires the data sent by v.

Problem: communication costs



$$c(v_1) = 2$$
$$c(v_2) = 1$$
$$c(v_3) = 3$$
$$c(v_4) = 2$$
$$c(v_5) = 4$$

- The objective function is equivalent to minimizing maximum send volume (maxSV).

Part	**Send Vol.**	Rec. Vol.	Send+rec.
P_1	3 + 2	2 + 1	8
P_2	2 + 1 + 1	3 + 2 + 4	13
P_3	4	1	5
Total	13	13	26

- maxSRV ← max send+receive volume
- totV ← total communication volume

Hypergraph Model

- Hypergraph H = (V, N)
  - A net is a subset of vertices.
  - Each net n has cost $c_H(n)$
- We model the task graph G(V,E) as a hypergraph
  - For each task s in G, let v_s be the corresponding vertex in H.
  - For each task s in G, the net set N contains a net n_s.
    - $n_s = \{v_d : ((s,d) \in E)\} \bigcup \{v_s\}$
    - $c_H(n_s) = c(s)$

Hypergraph Model

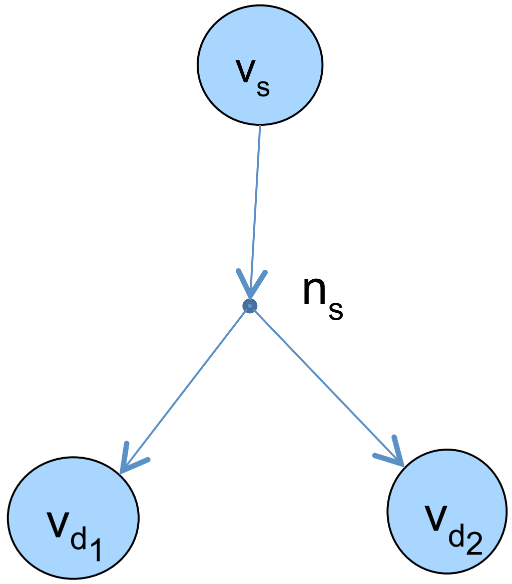
- λ_n: Connectivity of a net **n**, i.e., the number of parts net **n** is connected.

$$totV = \sum_{n \in N} c_H(n) \times (\lambda_n - 1)$$

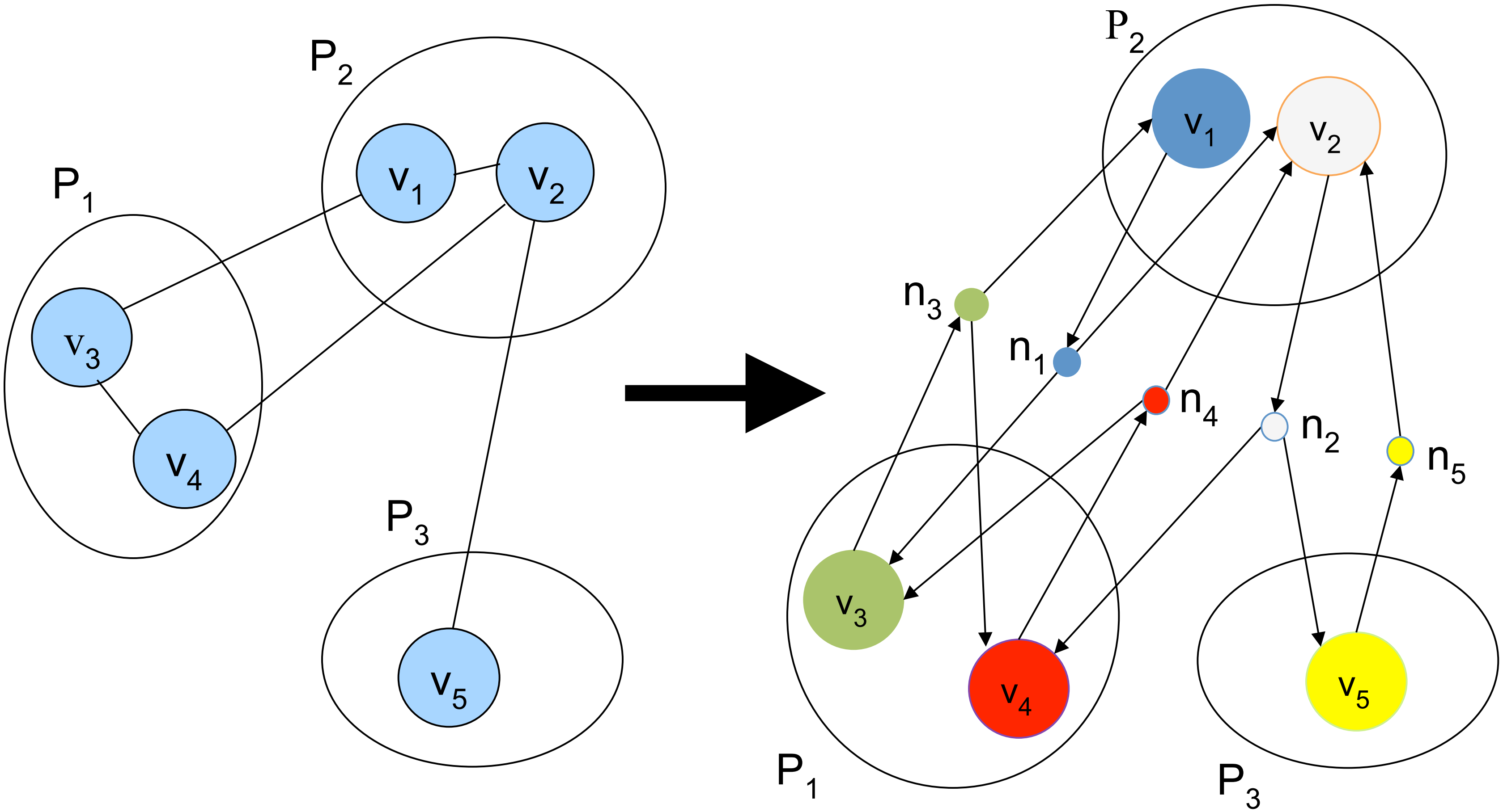
Minimizing the formula, equivalent to minimizing the total communication volume [Ç & Aykanat'99].

Directed Hypergraph Model

- Directed hypergraph:
  - Flow: from the source pin to the other pins.
  - Source of $n_s = v_s$
- Allows to minimize maxSV and maxSRV (in addition to totV).
- Objective: Partition the vertices into K parts s.t.
  - The load is distributed equally.
    - $W_k < W_{avg}(1+\varepsilon) \text{ for } 1 \le k \le K$
  - maxSV is minimized.
    - $SV(P) = \sum_{v_s \in P} c_H(n_s) \times (\lambda_{n_s} - 1)$
    - $\max SV = \max_k (SV(P_k))$



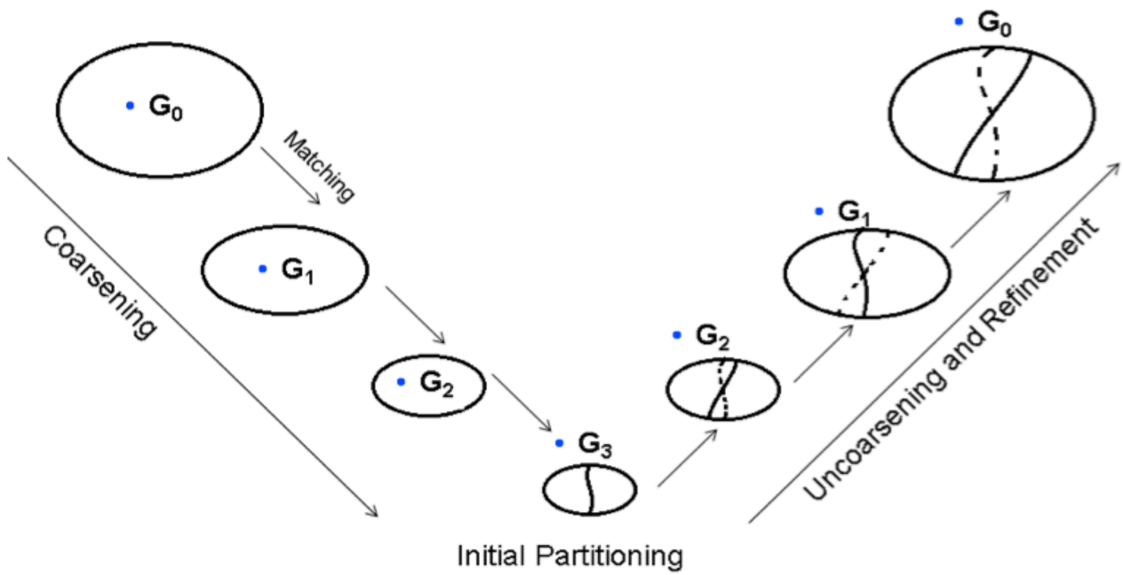
Hypergraph Model: Example



$$\lambda_{n_2} = 3,\ \lambda_{n_1} = \lambda_{n_3} = \lambda_{n_4} = \lambda_{n_5} = 2$$

Multi-level Approach

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



Multi-level Approach

- Most of the available tools adapt multi-level approach with **recursive bisection method.**
  - A partition is obtained by recursive partitioning into 2 parts.
  - Works fine for total communication.
  - May not be suitable for minimizing maximum send (and/or send+receive) volume.
    - Only the information about 2 parts is available at each step.
    - Send and receive volumes of other parts are unknown.

UMPa

- K-way multi-level partitioner.
- Uses directed hypergraph model.
  - Communication of the net flows from source to target vertices.
- Minimizes maxSV, while breaking ties by favoring reducing maxSRV, then the total volume.
- Currently, only coarsening phase of UMPa is shared memory parallel.
- Ultimate goal: To parallelize UMPa (MPI+OpenMP).

UMPa: Coarsening

- Neighbor vertices (u and v) are clustered by using agglomerative matching in coarsening phase.
  - Similarity of u and v

$$\sum_{n \in nets(u) \cap nets(v)} \frac{c_H(n)}{(pins(n)-1)}$$

```
Data: H = (V, N), rep[]
for each vertex u ∈ V in parallel do
    if CHECKANDLOCK(u) then
        if u is matched before then
            UNLOCK(u)
            continue
        adj_u ← {}
        for each vertex v ∈ neighbors[u] do
            adj_u ← adj_u ∪ {v}
            conn[v] ← sim(v,u)
        v* ← u
        conn* ← 0
        for each vertex v ∈ adj_u do
            v^r ← rep[v]
            if v^r ≠ v then
                conn[v^r] ← conn[v^r] + conn[v]
            if conn[v^r] > conn* then
                if CHECKANDLOCK(v^r) then
                    conn* ← conn[v^r]
                    v* ← v^r
                    UNLOCK(v*)
        rep[u] ← v*
        UNLOCK(v*)
        UNLOCK(u)
```

UMPa: Initial Partitioning

- For the initial partitioning, we used PaToH to obtain k initial parts.
  - Although PaToH is used to minimize total communication volume but not maximum send volume:
    - We do not want a drastic increase in any of the communication metrics. So, minimizing total volume is a good idea both in theory and practice.
    - Using recursive bisection and FM-based improvement are favorable due to the small net sizes and high vertex degrees.

UMPa: Uncoarsening

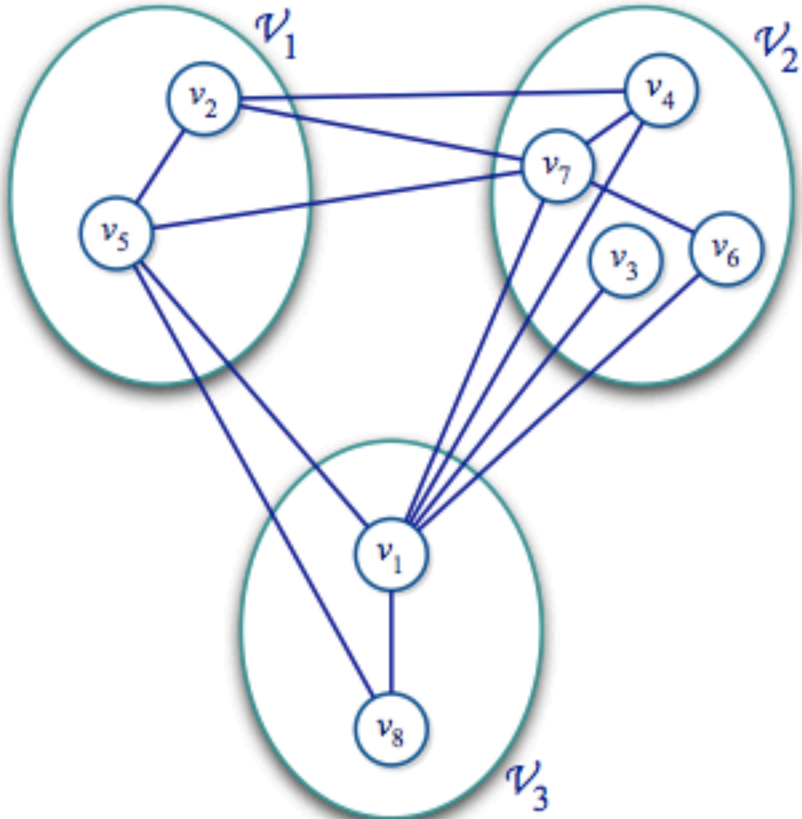
- Solutions for the coarser hypergraphs are iteratively projected to finer ones and refined.
- Refinement method:
  - Traverses the boundary vertices in random order.
    - Random, since FM/KL based heuristics are expensive especially in K-way.
  - Computes move gains for each visited vertex and select the best move.

```
Data: H = (V, N), boundary[], part[], SV[], SRV[]
for each unlocked u ∈ boundary do
    (bestMaxSV, bestMaxSRV, bestTotV) ← (maxSV, maxSRV, totV)
    bestPart ← part[u]
    for each part p other than part[u] do
        if p has enough space for vertex u then
            (SV[], SRV[], moveV) ←calculateComVols(v, p)
            (moveMaxSV, moveMaxSRV) ←
                            calculateMax(moveSV[], moveSRV[])
            MOVESELECT(moveMaxSV, moveMaxSRV, moveV, p,
                            bestMaxSV, bestMaxSRV, bestTotV, bestPart)
    if bestPart ≠ part[u] then
        move u to bestPart and update data structures accordingly
```

Move Selection

- Always move a visited vertex to the part with the maximum reduction on maxSV.
  - Tie-breaking is applied for equal reductions.
  - When there is an equality, the vertex move is favored toward the part that minimizes maxSRV, then totV.

Tie-breaking



Vertex	Part	*maxSV*	*maxSRV*	*totV*
v_1	$\mathcal{V}_1$	−1	+1	−2
v_1	$\mathcal{V}_2$	−2	−2	−3
v_2	$\mathcal{V}_2$	0	−1	−1
v_2	$\mathcal{V}_3$	−1	+1	0
v_3	$\mathcal{V}_1$	−1	−1	0
v_3	$\mathcal{V}_3$	−1	−1	−1
v_4	$\mathcal{V}_1$	−1	−1	0
v_4	$\mathcal{V}_3$	−1	+1	+1
v_5	$\mathcal{V}_3$	0	0	−1
v_6	$\mathcal{V}_1$	−1	0	+1
v_6	$\mathcal{V}_3$	−1	0	0
v_7	$\mathcal{V}_1$	−1	+1	0
v_7	$\mathcal{V}_3$	−1	−1	0
v_5	$\mathcal{V}_2$	+2	+2	−1
v_8	$\mathcal{V}_1$	0	0	0
v_8	$\mathcal{V}_2$	+2	+2	+1

- Initially, maxSV=6, maxSRV=9, totV= 12. (v_3, v_4, v_6, v_7)

Experimental Results

- Experiments
  - 123 graphs
  - 10 graph classes
  - For K = 4, 16, 64, 256
  - Each instance is partitioned 10 times.
- Compared with PaToH minimizing total volume.
- The power of tie-breaking is also studied.

Average Performance

K	PaToH + Refinement No tie breaking			UMPa No tie breaking			UMPa With tie breaking		
	$maxSV$	$maxSRV$	$totV$	$maxSV$	$maxSRV$	$totV$	$maxSV$	$maxSRV$	$totV$
4	0.93	1.05	1.06	0.73	0.83	0.93	0.66	0.77	0.84
16	0.93	1.06	1.04	0.84	0.94	1.11	0.73	0.83	0.98
64	0.91	1.04	1.02	0.86	0.98	1.12	0.76	0.87	1.00
256	0.91	1.03	1.01	0.89	1.00	1.10	0.81	0.91	1.02
Avg.	0.92	1.05	1.03	0.83	0.93	1.06	0.74	0.84	0.96

- The geometric mean of the relative results wrt PaToH used to minimize totV.
- Tie-breaking is very useful.
- As K increases the reduction rate decreases, since the total communication is distributed to more parts.

Medical Center

Experiment Results: K=16

Graph	$maxSV$	$maxSRV$	$totV$	Time
coPapersDBLP	0.862	0.845	1.252	1.591
as-22july06	0.760	0.787	1.016	4.286
road_central	0.558	0.577	0.716	0.247
smallworld	0.907	0.909	0.928	6.236
delaunay_n14	0.966	1.004	1.019	4.632
delaunay_n17	0.917	0.928	1.033	2.330
hugetrace-00010	0.980	0.981	1.107	0.462
hugetric-00020	0.964	0.964	1.075	0.484
venturiLevel3	0.924	0.925	1.072	0.584
adaptive	0.944	0.945	1.062	0.543
rgg_n_2_15_s0	0.815	0.867	0.982	3.029
rgg_n_2_21_s0	0.919	0.949	1.030	0.440
tn2010	0.838	1.062	4.214	1.222
ut2010	0.219	0.253	0.328	1.907
af_shell9	0.987	0.986	1.065	0.583
audikw1	0.787	0.826	1.094	0.479
asia.osm	0.476	0.496	0.790	0.190
belgium.osm	0.855	0.865	0.990	0.408
memplus	0.696	0.522	1.267	3.130
t60k	0.958	0.958	1.055	3.414

- Best and worst improvements for each graph class normalized w.r.t. PaToH.
- 78% (75%, 67%) improvement on maxSV for ut2010.

Experiment Results: K=256

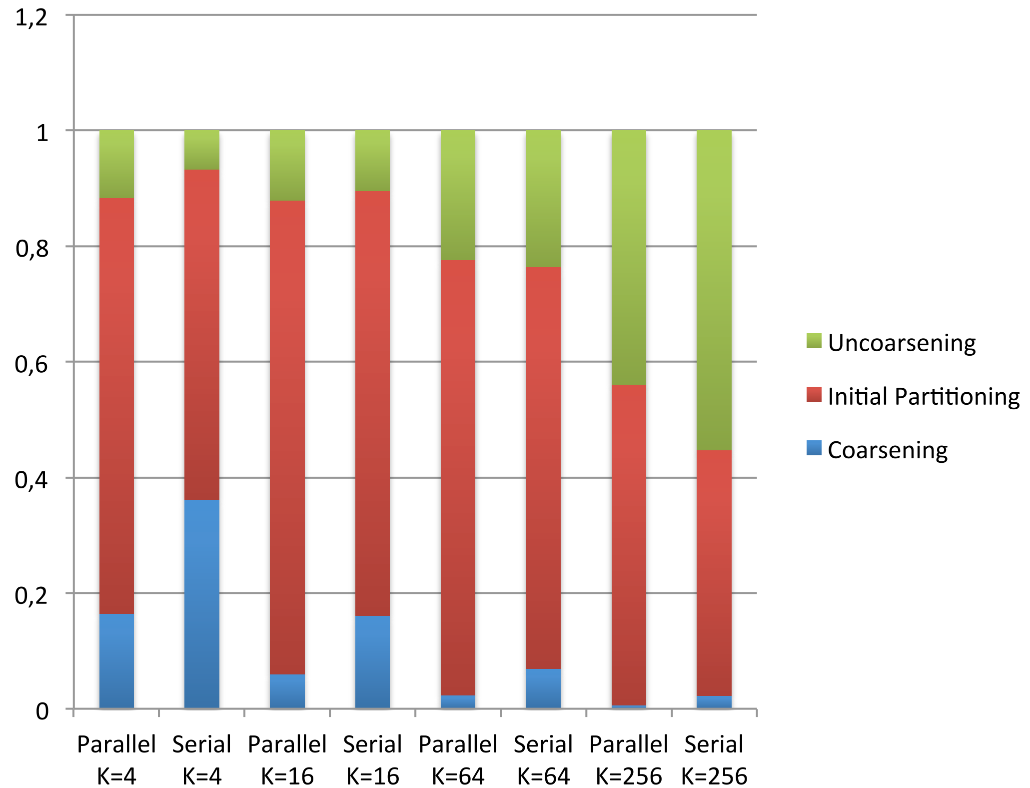
Graph	$maxSV$	$maxSRV$	$totV$	Time
coPapersCiteseer	0.694	0.693	1.005	2.937
coPapersDBLP	0.730	0.690	0.972	7.216
as-22july06	0.397	0.637	1.181	12.495
smallworld	0.839	0.843	0.899	7.965
delaunay_n20	0.927	0.943	1.024	3.829
delaunay_n21	0.948	0.963	1.033	3.066
hugetrace-00000	1.020	1.021	1.075	2.228
hugetric-00010	0.950	0.951	1.065	1.814
adaptive	0.976	0.978	1.063	2.322
venturiLevel3	0.993	0.996	1.057	2.642
rgg_n_2_22_s0	0.906	0.941	1.010	1.647
rgg_n_2_23_s0	0.862	0.891	1.009	1.286
ri2010	0.866	0.965	0.994	12.028
tx2010	0.586	0.816	0.951	1.836
af_shell10	0.986	0.987	1.056	1.758
audikw1	0.895	0.917	1.018	2.551
asia.osm	0.917	0.925	0.989	0.260
great-britain.osm	0.788	0.804	0.997	0.505
finan512	0.965	1.040	1.022	10.073
memplus	0.509	0.541	1.264	16.837

- 50% improvement on maxSV and maxSRV for memplus although total volume increases by 26%.

Execution Time

- K-way partitioners are costly due to the complexity of the refinement heuristic for maxSV.
- UMPa gets slower when the number of parts is large

K	4	16	64	256	Avg.
Relative time	1.02	1.29	2.01	5.76	1.98



Conclusion & Future Work

- Proposed a directed hypergraph model to minimize maxSV, maxSRV and totV.
- We developed a multi-level, K-way partitioner, UMPa.
- Employed a tie-breaking scheme to handle multiple communication metrics.
- Currently, UMPa is parallel (shared memory) at coarsening phase.
- Parallelizing & speeding up UMPa and the proposed refinement approach.

Thanks

- For more information visit
  - http://bmi.osu.edu/hpc

- Research at the HPC Lab is funded by

NSF, Department of Energy, National Institutes of Health, National Cancer Institute, DAGSI The Dayton Area Graduate Studies Institute, Sandia National Laboratories, INL Idaho National Laboratory

for each $n \in \mathsf{nets}[u]$ **do**
  if $s(n) = u$ **then**
    $\mathsf{sendGain}[\mathsf{part}[u]] \leftarrow \mathsf{sendGain}[\mathsf{part}[u]] + (\lambda_n - 1)\mathsf{c}[n]$
    if $\Lambda(n, \mathsf{part}[u]) > 1$ **then**
      $receiveGain \leftarrow receiveGain - \mathsf{c}[n]$
      $uToPartU \leftarrow uToPartU + \mathsf{c}[n]$
  else if $\Lambda(n, \mathsf{part}[u]) = 1$ **then**
    $\mathsf{sendGain}[\mathsf{part}[s(n)]] \leftarrow \mathsf{sendGain}[\mathsf{part}[s(n)]] + \mathsf{c}[n]$
    $receiveGain \leftarrow receiveGain + \mathsf{c}[n]$

```
for each part p other than part[u] do
    if p has enough space for vertex u then
        receiveLoss ← 0
        sendLoss[] ← 0
        sendLoss[p] ← sendGain[part[u]] + uToPartU
        for each n ∈ nets[u] do
            if s(n) = u then
                if Λ(n,p) > 0 then
                    sendLoss[p] ← sendLoss[p] − c[n]
                    receiveLoss ← receiveLoss − c[n]
            else if Λ(n,p) = 0 then
                sendLoss[part[s(n)]] ← sendLoss[part[s(n)]] + c[n]
                receiveLoss ← receiveLoss + c[n]
        (moveSV, moveSRV) ← (−∞, −∞)
        for each part q do
            Δ_S ← sendLoss[q] − sendGain[q]
            Δ_R ← 0
            if q = part[u] then
                Δ_R ← receiveGain
            else if q = p then
                Δ_R ← receiveLoss
            moveSV ← max(moveSV, SV[q] + Δ_S)
            moveSRV ← max(moveSRV, SV[q] + Δ_S + RV[q] + Δ_R)
        moveV ← totV + receiveLoss − receiveGain
        MoveSelect(moveSV, moveSRV, moveV, p,
                   bestMaxSV, bestMaxSRV, bestTotV, bestPart)
```