Recent Advances in Shared-Memory Hypergraph Partitioning

February 25, 2022
Lars Gottesbüren, Tobias Heuer, Peter Sanders, Sebastian Schlag
Hypergraphs

- generalization of graphs
  ⇒ hyperedges connect ≥ 2 nodes

- graphs ⇒ dyadic (2-ary) relationships

- hypergraphs ⇒ (d-ary) relationships

- hypergraph $H = (V, E, c, \omega)$
  - vertex set $V = \{1, \ldots, n\}$
  - edge set $E \subseteq \mathcal{P}(V) \setminus \emptyset$
  - node weights $c : V \to \mathbb{R}_{\geq 1}$
  - edge weights $\omega : E \to \mathbb{R}_{\geq 1}$
\( \varepsilon \)-Balanced Hypergraph Partitioning Problem

Partition hypergraph \( H = (V, E, c, \omega) \) into \( k \) disjoint blocks \( \Pi = \{ V_1, \ldots, V_k \} \) such that:

- blocks \( V_i \) are roughly equal-sized:

\[
c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil
\]
**ε-Balanced Hypergraph Partitioning Problem**

Partition hypergraph $H = (V, E, c, \omega)$ into $k$ disjoint blocks $\Pi = \{ V_1, \ldots, V_k \}$ such that:

- blocks $V_i$ are **roughly equal-sized**:
  \[ c(V_i) \leq (1 + \epsilon) \left\lceil \frac{c(V)}{k} \right\rceil \]

- connectivity objective is minimized.
**ε-Balanced Hypergraph Partitioning Problem**

Partition hypergraph \( H = (V, E, c, \omega) \) into \( k \) disjoint blocks \( \Pi = \{ V_1, \ldots, V_k \} \) such that:

- Blocks \( V_i \) are **roughly equal-sized**:
  \[
  c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil
  \]

- **Connectivity** objective is **minimized**.
ε-Balanced Hypergraph Partitioning Problem

Partition hypergraph $H = (V, E, c, \omega)$ into $k$ disjoint blocks $\Pi = \{V_1, \ldots, V_k\}$ such that:

- blocks $V_i$ are roughly equal-sized:
  \[ c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil \]

- connectivity objective is minimized:
  \[ \sum_{e \in E} (\lambda(e) - 1) \omega(e) = 12 \]
Applications

Distributed Databases

Route Planning

VLSI Design

HPC
Trade-Off Landscape
Trade-Off Landscape

- **Speed** (slow to fast)
- **Quality** (low to high)

- Sequential
- Shared Memory
- Distributed

- Tools:
  - PaToH-Q
  - PaToH-D
  - hMetis-R
  - Mt-KaHyPar-D [ALENEX'21] [with 10 threads]
  - Zoltan
  - BiPart
  - Social Hash
  - KaHyPar-CA
  - KaHyPar-HFC
Trade-Off Landscape

- KaHyPar-HFC
- KaHyPar-CA
- hMetis-R
- Mt-KaHyPar-Q [ALENEX'22] [with 10 threads]
- Mt-KaHyPar-D [ALENEX'21] [with 10 threads]
- PaToH-Q
- PaToH-D
- Zoltan
- BiPart
- Social Hash

Sequential
Shared Memory
Distributed

Speed

Quality

low

high
Trade-Off Landscape

KaHyPar-HFC
KaHyPar-CA
hMetis-R
Mt-KaHyPar-Q-F [submitted to SEA’22] [with 10 threads]
Mt-KaHyPar-Q [ALENEX’22] [with 10 threads]
Mt-KaHyPar-D [ALENEX’21] [with 10 threads]
PaToH-Q
PaToH-D
Zoltan
BiPart
Social Hash

Sequential
Shared Memory
Distributed
Multilevel Partitioning

Input Hypergraph

Coarsening

cluster

contract

...
Multilevel Partitioning

Input Hypergraph

Coarsening

Cluster

Contract

Initial Partitioning
Multilevel Partitioning

Coarsening

Input Hypergraph

contract

cluster

Initial Partitioning

Uncoarsening

local search

uncontract
Mt-KaHyPar: Algorithmic Components

Coarsening

Input Hypergraph

cluster

contract

Initial Partitioning

Uncoarsening

local search

uncontract
Mt-KaHyPar: Algorithmic Components

Parallel Coarsening
- Traditional \(\log(n)\)-level Coarsening (Mt-KaHyPar-D)
- \(n\)-level Coarsening (Mt-KaHyPar-Q)

Initial Partitioning → Uncoarsening

Thread 1 → Thread 2

Input Hypergraph

Local search
Uncontract

Mt-KaHyPar: Algorithmic Components
Mt-KaHyPar: Algorithmic Components

Parallel Coarsening

Traditional log(n)-level Coarsening (Mt-KaHyPar-D)

n-level Coarsening (Mt-KaHyPar-Q)

Parallel Recursive Bipartitioning based Initial Partitioning with Work-Stealing

$k = 4$

Parallel Recursion
Mt-KaHyPar: Algorithmic Components

Parallel Coarsening
- Traditional log(n)-level Coarsening (Mt-KaHyPar-D)
- n-level Coarsening (Mt-KaHyPar-Q)

Parallel Recursive Bipartitioning based Initial Partitioning with Work-Stealing

Parallel Direct k-Way FM
- Moves vertices greedily

Parallel Flow-Based Refinement
- Moves vertices greedily

Traditional Multilevel Partitioning

- contracts matching or clustering on each level
Traditional Multilevel Partitioning

- contracts matching or clustering on each level
Traditional Multilevel Partitioning

- contracts matching or clustering on each level
Traditional Multilevel Partitioning

- contracts matching or clustering on each level
Traditional Multilevel Partitioning

- contracts matching or clustering on each level
Traditional Multilevel Partitioning

- contracts matching or clustering on each level

$\Rightarrow$ approximately $O(\log n)$ levels
Traditional Multilevel Partitioning

Contracts matching or clustering on each level

\[ \Rightarrow \text{approximately } \mathcal{O}(\log n) \text{ levels} \]
\(n\)-level Partitioning

- contract one vertex at a time
**n-level Partitioning**

- contract one vertex at a time

![Diagram of n-level Partitioning](image)
\( n \)-level Partitioning

- contract one vertex at a time
**n-level Partitioning**

- contract one vertex at a time

![Diagram showing n-level Partitioning process with vertices and edges](image-url)
n-level Partitioning

- contract one vertex at a time
\(n\)-level Partitioning

- contract one vertex at a time

**Coarsening**: Almost \(n\) levels

**Unoarsening**: Almost \(n\) local search invocations \(\Rightarrow\) **High Quality**! (used in KaHyPar)
\( n \)-level Partitioning

- contract one vertex at a time

\[ \begin{align*}
\text{Coarsening:} & \text{ Almost } n \text{ levels} \\
\text{Unoarsening:} & \text{ Almost } n \text{ local search invocations } \Rightarrow \text{High Quality!} \text{ (used in KaHyPar)}
\end{align*} \]

\( \Rightarrow \text{Inherently Sequential!} \)
Contraction Forest

Any sequence of contractions form a forest
Contraction Forest

Any sequence of contractions form a forest
Contraction Forest

Any sequence of contractions form a forest

$v_4$ is contracted onto $v_2$
Contraction Forest

Any sequence of contractions form a forest

Contraction Forest

Roots are the vertices of the coarsest hypergraph
Any sequence of contractions form a forest

Contraction order:
1. Contract $v_{15}$ onto $v_8$
2. Contract $v_8$ onto $v_4$
3. Contract $v_4$ onto $v_2$
Contraction Forest

Any sequence of contractions form a forest

Contraction Forest

Observations

- There is more than one contraction order leading to the same contraction forest
Contraction Forest

Any sequence of contractions form a forest

Observations
- There is more than one contraction order leading to the same contraction forest

Rules
- Contractions in different subtrees are independent
- Contract $v$ when its children are contracted onto $v$
Contraction Forest

Any sequence of contractions form a forest

Observations
- There is more than one contraction order leading to the same contraction forest

Rules
- Contractions in different subtrees are independent
- Contract \( v \) when its children are contracted onto \( v \)

Parallelization Idea
- Contract contraction forest bottom-up in parallel
Contraction Forest

Any sequence of contractions form a forest

Observations
- There is more than one contraction order leading to the same contraction forest

Rules
- Contractions in different subtrees are independent
- Contract \( v \) when its children are contracted onto \( v \)

Parallelization Idea
- Contract contraction forest bottom-up in parallel
Contraction Forest

Any sequence of contractions form a forest

Observations
- There is more than one contraction order leading to the same contraction forest

Rules
- Contractions in different subtrees are independent
- Contract $v$ when its children are contracted onto $v$

Parallelization Idea
- Contract contraction forest bottom-up in parallel
Contraction Forest

Any sequence of contractions form a forest

\[ T_i = \text{Thread } i \]

Observations

- There is more than one contraction order leading to the same contraction forest

Rules

- Contractions in different subtrees are independent
- Contract \( v \) when its children are contracted onto \( v \)

Parallelization Idea

- Contract contraction forest bottom-up in parallel
Contraction Forest

Any sequence of contractions form a forest

Contraction Forest

\[ \bullet v_1 \quad \bullet v_2 \]

Observations

- There is more than one contraction order leading to the same contraction forest

Rules

- Contractions in different subtrees are independent
- Contract \( v \) when its children are contracted onto \( v \)

Parallelization Idea

- Contract contraction forest bottom-up in parallel
Contraction Forest

Any sequence of contractions form a forest

Contraction Forest
\[ \bullet v_1 \bullet v_2 \]

Observations
- There is more than one contraction order leading to the same contraction forest

Rules
- Contractions in different subtrees are independent
- Contract \( v \) when its children are contracted onto \( v \)

Parallelization Idea
- Contract contraction forest bottom-up in parallel

Problem: Contraction forest is not known in advance
Contraction Forest Construction

**Idea**: Construct contraction forest *on-the-fly*

### Algorithm 1: Parallel n-level Coarsening

```plaintext
for each \( u \in V \) in parallel
    \( v \leftarrow \text{find contraction partner for } u \)
    if add \( (v, u) \) to contraction forest then
        contract \( v \) onto \( u \)
```

Contraction Forest Construction

Idea: Construct contraction forest on-the-fly

Algorithm 1: Parallel n-level Coarsening

for each $u \in V$ in parallel

$v \leftarrow \text{find contraction partner for } u$

if add $(v, u)$ to contraction forest then

contract $v$ onto $u$
Contraction Forest Construction

Idea: Construct contraction forest \textit{on-the-fly}

\begin{algorithm}
\noindent for each \( u \in V \) in parallel
\noindent \hspace{1em} \( v \leftarrow \text{find contraction partner for } u \) \\
\noindent \hspace{1em} \textbf{if} \hspace{1em} \text{add } (v, u) \text{ to contraction forest } \textbf{then} \\
\noindent \hspace{2em} \text{contract } v \text{ onto } u
\end{algorithm}

\[ T_i = \text{Thead } i \]
Contraction Forest Construction

Idea: Construct contraction forest on-the-fly

Algorithm 1: Parallel $n$-level Coarsening

\begin{verbatim}
for each $u \in V$ in parallel
    $v \leftarrow$ find contraction partner for $u$
    if add $(v, u)$ to contraction forest then
        contract $v$ onto $u$
\end{verbatim}
Contraction Forest Construction

**Idea:** Construct contraction forest *on-the-fly*

\[ T_i = \text{Thead } i \]

### Algorithm 1: Parallel $n$-level Coarsening

```plaintext
for each $u \in V$ in parallel
    $v \leftarrow$ find contraction partner for $u$
    if add $(v, u)$ to contraction forest then
        contract $v$ onto $u$
```

![Diagram](image.png)
Contraction Forest Construction

**Idea:** Construct contraction forest *on-the-fly*

$T_i = \text{Thead } i$

**Algorithm 1:** Parallel $n$-level Coarsening

```plaintext
for each $u \in V$ in parallel
    $v \leftarrow$ find contraction partner for $u$
    if add $(v, u)$ to contraction forest then
        contract $v$ onto $u$
```

Contraction Forest Construction

Idea: Construct contraction forest \textit{on-the-fly}

(v_2, v_3) is not eligible for contraction
\implies do something else

\[ T_i = \text{Thead } i \]

\textbf{Algorithm 1: Parallel } n\text{-level Coarsening}

\begin{algorithm}
\textbf{for each } u \in V \text{ in parallel }
\begin{align*}
&v \leftarrow \text{find contraction partner for } u \\
&\text{if add } (v, u) \text{ to contraction forest then} \\
&\quad \text{contract } v \text{ onto } u
\end{align*}
\end{algorithm}
Contraction Forest Construction

**Idea:** Construct contraction forest *on-the-fly*

$$T_i = \text{Thead } i$$

Algorithm 1: Parallel $n$-level Coarsening

for each $u \in V$ in parallel

v ← find contraction partner for $u$

if add ($v$, $u$) to contraction forest then

contract $v$ onto $u$
Contraction Forest Construction

Idea: Construct contraction forest on-the-fly

Algorithm 1: Parallel n-level Coarsening

for each $u \in V$ in parallel

$v \leftarrow$ find contraction partner for $u$

if add $(v, u)$ to contraction forest then

contract $v$ onto $u$

$T_i = \text{Thead } i$
Contraction Forest Construction

Idea: Construct contraction forest on-the-fly

Algorithm 1: Parallel $n$-level Coarsening

```plaintext
for each $u \in V$ in parallel
    $v \leftarrow$ find contraction partner for $u$
    if add $(v, u)$ to contraction forest then
        contract $v$ onto $u$
```

$T_i = \text{Thead } i$
Contraction Forest Construction

Idea: Construct contraction forest on-the-fly

Algorithm 1: Parallel n-level Coarsening

for each $u \in V$ in parallel

$\quad v \leftarrow \text{find contraction partner for } u$

$\quad \text{if add } (v, u) \text{ to contraction forest then}$

$\quad \text{contract } v \text{ onto } u$

Cyclic Dependency
$\Rightarrow$ Discard Contraction
Contraction Forest Construction

Idea: Construct contraction forest on-the-fly

Algorithm 1: Parallel $n$-level Coarsening

for each $u \in V$ in parallel

\[
\begin{align*}
    v &\leftarrow \text{find contraction partner for } u \\
    \text{if } & \text{add } (v, u) \text{ to contraction forest} \text{ then} \\
    & \text{contract } v \text{ onto } u
\end{align*}
\]

$T_i = \text{Thead } i$

Pending counter of $v_8$ is zero

$\Rightarrow$ we assume contraction of $v_8$ has already started

$\Rightarrow$ find suitable ancestor of $v_8$
Contraction Forest Construction

**Idea:** Construct contraction forest *on-the-fly*

\[ T_i = \text{Thead } i \]

**Algorithm 1:** Parallel \( n \)-level Coarsening

```
for each \( u \in V \) in parallel
  \( v \leftarrow \text{find contraction partner for } u \)
  if \( \text{add } (v, u) \) to contraction forest then
    contract \( v \) onto \( u \)
```
Contraction Forest Construction

Idea: Construct contraction forest on-the-fly

\[ T_i = \text{Thead } i \]

Algorithm 1: Parallel n-level Coarsening

for each \( u \in V \) in parallel

\[ v \leftarrow \text{find contraction partner for } u \]

if add \((v, u)\) to contraction forest then

contract \( v \) onto \( u \)
Contraction Forest Construction

**Idea:** Construct contraction forest *on-the-fly*

Thread $T_3$ decreases pending counter of $v_2$ to zero
⇒ Recursively continue

Algorithm 1: Parallel $n$-level Coarsening

```plaintext
for each $u \in V$ in parallel
    $v \leftarrow$ find contraction partner for $u$
    if add $(v, u)$ to contraction forest then
        contract $v$ onto $u$
```
Contraction Forest Construction

Idea: Construct contraction forest on-the-fly

\[ T_i = \text{Thead } i \]

Algorithm 1: Parallel $n$-level Coarsening

for each $u \in V$ in parallel

\[ v \leftarrow \text{find contraction partner for } u \]

if add $(v, u)$ to contraction forest then

\[ \text{contract } v \text{ onto } u \]
Contraction Forest Construction

**Idea**: Construct contraction forest *on-the-fly*

\[ T_i = \text{Thead } i \]

**Algorithm 1**: Parallel \( n \)-level Coarsening

\[
\begin{align*}
\text{for each } u \in V \text{ in parallel} \\
& v \leftarrow \text{find contraction partner for } u \\
& \text{if } \text{add}(v, u) \text{ to contraction forest} \text{ then} \\
& \quad \text{contract } v \text{ onto } u
\end{align*}
\]
Contraction Forest Construction

Idea: Construct contraction forest on-the-fly

Algorithm 1: Parallel n-level Coarsening

for each $u \in V$ in parallel

$v \leftarrow$ find contraction partner for $u$

if add $(v, u)$ to contraction forest then

contract $v$ onto $u$

Simple locking protocol used to modify contraction forest

$T_i = $ Thead $i$
Consistency Requirements

Contraction Consistency

Data Structure Consistency
Consistency Requirements

Contraction Consistency

Data Structure Consistency

see paper
Parallel Uncoarsening

- traditional $n$-level uncontracts only one vertex on each level $\Rightarrow$ inherently sequential
Parallel Uncoarsening

- traditional $n$-level uncontracts only one vertex on each level $\Rightarrow$ inherently sequential

Idea
- assemble independent uncontractions in a batch $B$ with $|B| \approx b_{\text{max}}$
- uncontract $B$ in parallel
- then run parallel localized refinement around $B$
- construct batches $B = \langle B_1, \ldots, B_l \rangle$
- uncontracting $B_i$ enables uncontraction of all vertices in $B_{i+1}$
Parallel Uncoarsening

- traditional $n$-level uncontracts only one vertex on each level $\Rightarrow$ inherently sequential

**Idea**
- assemble independent uncontractions in a batch $B$ with $|B| \approx b_{\text{max}}$
- uncontract $B$ in parallel
- then run parallel localized refinement around $B$
- construct batches $\mathcal{B} = \langle B_1, \ldots, B_l \rangle$
- uncontracting $B_i$ enables uncontraction of all vertices in $B_{i+1}$
- top-down traversal of contraction forest $\mathcal{F}$

$b_{\text{max}} = 3$

$\mathcal{B} = \langle \quad, \quad, \quad, \quad, \quad, \quad \rangle$
Parallel Uncoarsening

- traditional \( n \)-level uncontracts only one vertex on each level \(\Rightarrow\) inherently sequential

**Idea**
- assemble independent uncontractions in a *batch* \(B\) with \(|B| \approx b_{\text{max}}\)
- uncontract \(B\) in parallel
- then run parallel localized refinement around \(B\)
- construct *batches* \(\mathcal{B} = \langle B_1, \ldots, B_l \rangle\)
- uncontracting \(B_i\) enables uncontraction of all vertices in \(B_{i+1}\)
- *top-down traversal* of contraction forest \(\mathcal{F}\)

\[
\begin{align*}
    b_{\text{max}} &= 3 \\
    \mathcal{B} &= \langle \langle v_3, v_7, v_4 \rangle, \quad , \quad , \quad , \quad , \quad \rangle
\end{align*}
\]
Parallel Uncoarsening

- traditional \( n \)-level uncontracts only one vertex on each level \( \Rightarrow \) inherently sequential

**Idea**
- assemble independent uncontractions in a batch \( B \) with \( |B| \approx b_{\text{max}} \)
  - uncontract \( B \) in parallel
  - then run parallel localized refinement around \( B \)
- construct batches \( B = \langle B_1, \ldots, B_l \rangle \)
- uncontracting \( B_i \) enables uncontraction of all vertices in \( B_{i+1} \)
- **top-down traversal** of contraction forest \( \mathcal{F} \)

\[
b_{\text{max}} = 3
\]

\[
B = \langle \langle v_3, v_7, v_4 \rangle, \langle v_5, v_6, v_{12} \rangle, \text{eligible for uncontraction}, \text{eligible for uncontraction}, \text{already uncontracted} \rangle
\]
Parallel Uncoarsening

- traditional $n$-level uncontracts only one vertex on each level $\Rightarrow$ inherently sequential

Idea
- assemble independent uncontractions in a batch $B$ with $|B| \approx b_{\text{max}}$
  - uncontract $B$ in parallel
  - then run parallel localized refinement around $B$
- construct batches $\mathcal{B} = \langle B_1, \ldots, B_l \rangle$
- uncontracting $B_i$ enables uncontraction of all vertices in $B_{i+1}$
- top-down traversal of contraction forest $\mathcal{F}$

- $b_{\text{max}} = 3$
- $\mathcal{B} = \langle \{v_3, v_7, v_4\}, \{v_5, v_6, v_{12}\}, \{v_8, v_9, v_{10}\}, \ldots \rangle$

- eligible for uncontraction
- already uncontracted
Parallel Uncoarsening

- traditional \(n\)-level uncontracts only one vertex on each level \(\Rightarrow\) inherently sequential

Idea
- assemble independent uncontractions in a batch \(B\) with \(|B| \approx b_{\text{max}}\)
  - uncontract \(B\) in parallel
  - then run parallel localized refinement around \(B\)
- construct batches \(B = \langle B_1, \ldots, B_l \rangle\)
- uncontracting \(B_i\) enables uncontraction of all vertices in \(B_{i+1}\)
- top-down traversal of contraction forest \(\mathcal{F}\)

\[b_{\text{max}} = 3\]
\[
B = \langle V_3, V_7, V_4 , V_5, V_6, V_{12} , V_8, V_9, V_{10} , V_{11}, V_{13}, V_{14}, \rangle
\]
Parallel Uncoarsening

- traditional \( n \)-level uncontracts only one vertex on each level \( \Rightarrow \) inherently sequential

Idea
- assemble independent uncontractions in a batch \( B \) with \(|B| \approx b_{\text{max}}\)
- uncontract \( B \) in parallel
- then run parallel localized refinement around \( B \)
- construct batches \( B = \langle B_1, \ldots, B_l \rangle \)
- uncontracting \( B_i \) enables uncontraction of all vertices in \( B_{i+1} \)
- top-down traversal of contraction forest \( \mathcal{F} \)

\[ b_{\text{max}} = 3 \]
\[ B = \langle v_3, v_7, v_4, v_5, v_6, v_{12}, v_8, v_9, v_{10}, v_{11}, v_{13}, v_{14}, v_{15} \rangle \]
Parallel Uncoarsening

- traditional $n$-level uncontracts only one vertex on each level $\Rightarrow$ inherently sequential

**Idea**
- assemble independent uncontractions in a *batch* $B$ with $|B| \approx b_{\text{max}}$
- uncontract $B$ in parallel
- then run parallel localized refinement around $B$
- construct batches $B = \langle B_1, \ldots, B_l \rangle$
- uncontracting $B_i$ enables uncontraction of all vertices in $B_{i+1}$
- **top-down traversal** of contraction forest $\mathcal{F}$

$b_{\text{max}} = 3$
$B = \langle \{v_3, v_7, v_4\}, \{v_5, v_6, v_{12}\}, \{v_8, v_9, v_{10}\}, \{v_{11}, v_{13}, v_{14}\}, \{v_{15}\}\rangle$

$b_{\text{max}} = 1000$ in practice

- eligible for uncontraction
- already uncontracted
Parallel Uncoarsening

- traditional \( n \)-level uncontracts only one vertex on each level \( \Rightarrow \) inherently sequential

**Idea**
- assemble independent uncontractions in a *batch* \( B \) with \( |B| \approx b_{\text{max}} \)
- uncontract \( B \) in parallel
- then run parallel localized refinement around \( B \)
- construct batches \( B = \langle B_1, \ldots, B_l \rangle \)
- uncontracting \( B_i \) enables uncontraction of all vertices in \( B_{i+1} \)
- top-down traversal of contraction forest \( \mathcal{F} \)

\[ b_{\text{max}} = 3 \]
\[ B = \langle v_3, v_7, v_4, v_5, v_6, v_{12}, v_8, v_9, v_{10}, v_{11}, v_{13}, v_{14}, v_{15} \rangle \]

**Implementation Detail:**
Uncontract siblings in reverse order of contraction
\( \Rightarrow \) see paper

- already uncontracted
Experiments – Large Instances

- for comparison with fast partitioners: Zoltan, PaToH-D, Hype, BiPart
- for scaling experiments
- 1st gen Epyc Rome, 1 socket, 64 cores @ 2.0-3.35 Ghz, 1024 GB RAM

- 94 large hypergraphs: [publicly available]
  - SuiteSparse Matrix Collection 42
  - SAT Competition 2014 (3 representations) $14 \cdot 3 = 42$
  - DAC2012 VLSI Circuits 10
- Largest hypergraph $\approx 2$ billion pins

- $k \in \{2, 8, 16, 64\}$ with imbalance: $\varepsilon = 3\%$
- 5 random seeds
- 1, 4, 16, 64 threads
Experiments – Scalability
Experiments – Scalability

harmonic mean speedup of Mt-KaHyPar-Q:
- 3.7 with 4 threads
- 11.7 with 16 threads
- 22.6 with 64 threads

instances ≥ 100s:
- 3.7 with 4 threads
- 12.3 with 16 threads
- 25 with 64 threads
Experiments – Medium-Sized Instances

- for comparison with sequential partitioners: KaHyPar, hMetis, PaToH
- Intel Xeon Gold, 2 sockets, 20 cores @ 2.1 Ghz, 96 GB RAM

- 488 hypergraphs: [publicly available]
  - SuiteSparse Matrix Collection 184
  - SAT Competition 2014 (3 representations) 92·3 = 276
  - DAC2012 VLSI Circuits 10
  - ISPD98 18

- $k \in \{2, 4, 8, 16, 32, 64, 128\}$ with imbalance: $\varepsilon = 3\%$
- 10 random seeds
- 10 threads
Experiments – Connectivity Metric (Quality)

![Graph showing connectivity metric quality for different algorithms over a range of parameters.](image-url)
Experiments – Connectivity Metric (Quality)

\[ p_{\text{Algo}}(\tau) = \frac{|\{ I \in \mathcal{I} \mid \text{Algo}(I) \leq \tau \cdot \text{Best}(I)\}|}{|\mathcal{I}|} \]
Experiments – Connectivity Metric (Quality)

\[ p_{\text{Algo}}(\tau) = \left| \left\{ I \in \mathcal{I} \mid \text{Algo}(I) \leq \tau \cdot \text{Best}(I) \right\} \right| / |\mathcal{I}| \]

\[ \tau = 1 \iff \text{fraction of instances for which algorithm finds the best partition} \]

KaHyPar-CA \approx 50\%

Mt-KaHyPar-Q \approx 37\%

Mt-KaHyPar-D \approx 5\%
Experiments – Connectivity Metric (Quality)

\[ p_{\text{Algo}}(\tau) = \frac{|\{I \in \mathcal{I} \mid \text{Algo}(I) \leq \tau \cdot \text{Best}(I)\}|}{|\mathcal{I}|} \]
Experiments – Connectivity Metric (Quality)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Gmean t[s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mt-KaHyPar-D 10</td>
<td>0.95</td>
</tr>
<tr>
<td>PaToH-D</td>
<td>1.17</td>
</tr>
<tr>
<td>Mt-KaHyPar-Q 10</td>
<td>3.19</td>
</tr>
<tr>
<td>PaToH-Q</td>
<td>5.86</td>
</tr>
<tr>
<td>KaHyPar-CA</td>
<td>28.14</td>
</tr>
</tbody>
</table>
$p_{Algo}(\tau) = \frac{|\{I \in \mathcal{I} \mid Algo(I) \leq \tau \cdot Best(I)\}|}{|\mathcal{I}|}$

![Graph showing the fraction of instances as a function of $\tau$. The graph compares two algorithms: Mt-KaHyPar-Q 10 (red line) and KaHyPar-HFC (purple line). The x-axis represents $\tau$ ranging from 1 to $10^2 \Theta$, and the y-axis represents the fraction of instances ranging from 0.01 to 1.00.]
\[
p_{\text{Algo}}(\tau) = \frac{|\{I \in \mathcal{I} | \text{Algo}(I) \leq \tau \cdot \text{Best}(I)\}|}{|\mathcal{I}|}
\]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Gmean $t$ [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mt-KaHyPar-Q 10</td>
<td>3.19</td>
</tr>
<tr>
<td>KaHyPar-HFC</td>
<td>48.98</td>
</tr>
</tbody>
</table>

Fraction of instances

![Graph showing the relationship between $\tau$ and the fraction of instances for different algorithms.](image-url)
$p_{\text{Algo}}(\tau) = \frac{|\{ I \in \mathcal{I} \mid \text{Algo}(I) \leq \tau \cdot \text{Best}(I)\}|}{|\mathcal{I}|}$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Gmean t[s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mt-KaHyPar-Q 10</td>
<td>3.19</td>
</tr>
<tr>
<td><strong>Mt-KaHyPar-Q-F 10</strong></td>
<td><strong>5.08</strong></td>
</tr>
<tr>
<td>KaHyPar-HFC</td>
<td>48.98</td>
</tr>
</tbody>
</table>
\[ p_{\text{Algo}}(\tau) = \frac{\{I \in \mathcal{I} \mid \text{Algo}(I) \leq \tau \cdot \text{Best}(I)\}}{|\mathcal{I}|} \]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Gmean t[s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mt-KaHyPar-Q 10</td>
<td>3.19</td>
</tr>
<tr>
<td>Mt-KaHyPar-Q-F 10</td>
<td>5.08</td>
</tr>
<tr>
<td>KaHyPar-HFC</td>
<td>48.98</td>
</tr>
</tbody>
</table>
Conclusion

Mt-KaHyPar

- achieves the **same solution quality** as the highest quality sequential system in fast parallel code
- **order of magnitude faster** than its sequential counterparts with only 10 threads
- great speedups

https://github.com/kahypar/mt-kahypar
Conclusion

Mt-KaHyPar

- achieves the same solution quality as the highest quality sequential system in fast parallel code
- order of magnitude faster than its sequential counterparts with only 10 threads
- great speedups

Future Work

- Thesis!
- Multi-Objective and Multi-Constraint Partitioning
- Large $k$ Partitioning
- Fixed Vertices

https://github.com/kahypar/mt-kahypar
Parallel Flow-Based Refinement

The value of a maximum flow between two vertices $s$ and $t$ is equal with the minimum cut separating $s$ and $t$. 
Parallel Flow-Based Refinement

The value of a \textit{maximum flow} between two vertices \(s\) and \(t\) is equal with the \textit{minimum cut} separating \(s\) and \(t\).
Parallel Flow-Based Refinement

The value of a maximum flow between two vertices $s$ and $t$ is equal with the minimum cut separating $s$ and $t$.

Bipartition $\Pi = \{V_1, V_2\}$

Hypergraph

Cut Hyperedges
Parallel Flow-Based Refinement

The value of a **maximum flow** between two vertices $s$ and $t$ is equal with the **minimum cut** separating $s$ and $t$.

Initial Cut = 539, Target Imbalance = 3%
Parallel Flow-Based Refinement

The value of a **maximum flow** between vertices $s$ and $t$ is equal with the **minimum cut** separating $s$ and $t$.

Initial Cut = 539, Target Imbalance = 3%

Grow region around cut via BFS
Parallel Flow-Based Refinement

The value of a maximum flow between to vertices $s$ and $t$ is equal with the minimum cut separating $s$ and $t$.

Initial Cut = 539, Target Imbalance = 3%
Parallel Flow-Based Refinement

The value of a \textbf{maximum flow} between vertices \( s \) and \( t \) is equal with the \textbf{minimum cut} separating \( s \) and \( t \)

Initial Cut = 539, Target Imbalance = 3%

Compute a maximum \((s, t)\)-flow
Parallel Flow-Based Refinement

The value of a **maximum flow** between two vertices $s$ and $t$ is equal with the **minimum cut** separating $s$ and $t$.

Initial Cut = 539, Target Imbalance = 3%

Current Cut = 250, Current Imbalance = 15% **Imbalanced!**
Parallel Flow-Based Refinement

The value of a **maximum flow** between vertices $s$ and $t$ is equal with the **minimum cut** separating $s$ and $t$

Initial Cut = 539, Target Imbalance = 3%

Contract smaller cut onto its terminal plus one additional node

Piercing node ensure that we find a different cut in the next iteration

Current Cut = 250, Current Imbalance = 15% **Imbalanced!**
Parallel Flow-Based Refinement

The value of a **maximum flow** between two vertices $s$ and $t$ is equal to the **minimum cut** separating $s$ and $t$

Initial Cut = 539, Target Imbalance = 3%

Contract smaller cut onto its terminal plus one additional node

![Diagram showing parallel flow-based refinement with vertices $V_1$ and $V_2$, source $S$, and sink $t$. The diagram illustrates the initial cut and the target imbalance.](image-url)
Parallel Flow-Based Refinement

The value of a maximum flow between to vertices \( s \) and \( t \) is equal with the minimum cut separating \( s \) and \( t \)

Initial Cut = 539, Target Imbalance = 3%

Augment flow again to a maximum \((s, t)\)-flow
Parallel Flow-Based Refinement

The value of a **maximum flow** between two vertices $s$ and $t$ is equal to the **minimum cut** separating $s$ and $t$.

Initial Cut = 539, Target Imbalance = 3%

Current Cut = 498, Current Imbalance = 2.5%

Balanced!

Improvement = 539 − 498 = 41
Parallel Flow-Based Refinement

The value of a maximum flow between two vertices $s$ and $t$ is equal with the minimum cut separating $s$ and $t$.

New Cut $= 498$, New Imbalance $= 2.5\%$
Parallel Flow-Based Refinement

The value of a maximum flow between two vertices $s$ and $t$ is equal with the minimum cut separating $s$ and $t$

New Cut = 498, New Imbalance = 2.5%

Our implementation uses a parallel push-relabel maximum flow algorithm
Parallel Flow-Based Refinement

General Idea: Schedule parallel flow problems on adjacent block pairs
Parallel Flow-Based Refinement

**General Idea:** Schedule parallel flow problems on adjacent block pairs

Nodes of flow problems can overlap
Parallel Flow-Based Refinement

General Idea: Schedule parallel flow problems on adjacent block pairs

- Flow computation returns a sequence of moves
- What could possibly go wrong?

Nodes of flow problems can overlap
Parallel Flow-Based Refinement

**General Idea:** Schedule parallel flow problems on adjacent block pairs

- Flow computation returns a sequence of moves.
- What could possibly go wrong?
  - Applying the move sequence could violate the balance constraint.
Parallel Flow-Based Refinement

**General Idea:** Schedule parallel flow problems on adjacent block pairs

- Flow computation returns a sequence of moves
- What could possibly go wrong?
  - Applying the move sequence could violate the balance constraint
  - Applying the move sequence could worsen the solution quality
Experiments – Connectivity Metric (Quality)
Experiments – Connectivity Metric (Quality)

\[ p_{\text{Algo}}(\tau) = \frac{|\{I \in \mathcal{I} \mid \text{Algo}(I) \leq \tau \cdot \text{Best}(I)\}|}{|\mathcal{I}|} \]

![Graph showing the fraction of instances as a function of \( \tau \)]
Experiments – Connectivity Metric (Quality)

\[ p_{\text{Algo}}(\tau) = \frac{|\{ I \in \mathcal{I} \mid \text{Algo}(I) \leq \tau \cdot \text{Best}(I)\}|}{|\mathcal{I}|} \]

\( \tau = 1 \Leftrightarrow \) fraction of instances for which algorithm finds the best partition

\( \text{Mt-KaHyPar-Q} \geq 70\% \)
Experiments – Connectivity Metric (Quality)

\[ p_{Algo}(\tau) = \frac{\left| \{ I \in \mathcal{I} \mid Algo(I) \leq \tau \cdot Best(I) \} \right|}{|\mathcal{I}|} \]

\( \tau = 1 \Leftrightarrow \) fraction of instances for which algorithm finds the best partition

- Mt-KaHyPar-Q \( \geq 70\% \)
- Zoltan \( \leq 5\% \)

\( \tau \)
- Mt-KaHyPar-Q 64
- PaToH-D
- Mt-KaHyPar-D 64
- Hype
- Zoltan 64
- BiPart 64
Experiments – Connectivity Metric (Quality)

$$p_{Algo}(\tau) = \frac{\{|I \in \mathcal{I} | Algo(I) \leq \tau \cdot Best(I)\}}{\mathcal{I}}$$

$\tau = 1.1 \iff$ fraction for which algorithm performs at most 10% worse than the best
Experiments – Connectivit### Connectivity Metric (Quality)

\[ p_{\text{Algo}}(\tau) = \frac{|\{ I \in \mathcal{I} \mid \text{Algo}(I) \leq \tau \cdot \text{Best}(I)\}|}{|\mathcal{I}|} \]
Experiments – Connectivity Metric (Quality)

\[ p_{\text{Algo}}(\tau) = \frac{|\{ I \in \mathcal{I} \mid \text{Algo}(I) \leq \tau \cdot \text{Best}(I)\}|}{|\mathcal{I}|} \]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Gmean</th>
<th>t[s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mt-KaHyPar-D 64</td>
<td></td>
<td>4.89</td>
</tr>
<tr>
<td>Zoltan 64</td>
<td></td>
<td>12.63</td>
</tr>
<tr>
<td>Hype</td>
<td></td>
<td>25.56</td>
</tr>
<tr>
<td>BiPart 64</td>
<td></td>
<td>29.19</td>
</tr>
<tr>
<td>Mt-KaHyPar-Q 64</td>
<td></td>
<td>30.7</td>
</tr>
<tr>
<td>PaToH-D</td>
<td></td>
<td>51.2</td>
</tr>
</tbody>
</table>