n-Level Hypergraph Partitioning

Vitali Henne, Henning Meyerhenke, Peter Sanders,
Sebastian Schlag, Christian Schulz

Karlsruhe Institute of Technology (KIT), 76128 Karlsruhe, Germany
{meyerhenke,sanders,sebastian.schlag,christian.schulz}@kit.edu
vitali.henne@gmail.com

Abstract. We develop a multilevel algorithm for hypergraph partitioning that contracts the vertices one at a time and thus allows very high quality. This includes a rating function that avoids nonuniform vertex weights, an efficient "semi-dynamic" hypergraph data structure, a very fast coarsening algorithm, and two new local search algorithms. One is a k-way hypergraph adaptation of Fiduccia-Mattheyses local search and gives high quality at reasonable cost. The other is an adaptation of size-constrained label propagation to hypergraphs. Comparisons with hMetis and PaToH indicate that the new algorithm yields better quality over several benchmark sets and has a running time that is comparable to hMetis. Using label propagation local search is several times faster than hMetis and gives better quality than PaToH for a VLSI benchmark set.

Keywords: hypergraph partitioning, local search, label propagation

1 Introduction

Context. Hypergraph partitioning (HGP) is an important problem with many application areas. Two prominent areas are VLSI design and scientific computing (e. g. the acceleration of sparse matrix-vector multiplications) [20]. While the former is an example of a field where small optimizations can lead to significant savings, the latter is an example where hypergraph-based modeling better captures the objectives of the application domain [8] than graph-based approaches. We focus on a version of the problem that partitions the vertices of a given hypergraph into k blocks of roughly equal size (in our case $1 + \varepsilon$ times the average block size) while optimizing an objective function. In this paper, we minimize the total cut size, i.e., the number of hyperedges that span multiple blocks.

Since the 1990s HGP has evolved into a broad research area [3,6,20]. The two most widely used general-purpose tools are PaToH [8] (originating from scientific computing) and hMetis [14,15] (originating from VLSI design). Other tools with certain distinguishing characteristics are known, in particular Mondriaan [26] (matrix partitioning), MLPart [2] (circuit partitioning), Zoltan [10] and Parkway [24] (parallel), and UMPa [25] (multi-objective). All these tools use the *multilevel paradigm*, which has three phases. The first of which recursively *coarsens* the hypergraph to obtain a hierarchy of smaller hypergraphs that reflect the basic structure of the input. After applying an *initial partitioning* algorithm to the smallest hypergraph in the second phase, coarsening is undone and, at each level, a *local search* method is used to improve the partition induced by the coarser level.

The two most popular local search approaches are greedy algorithms [15,25] or variations of the Fiduccia-Mattheyses (FM) heuristic [11]. FM-type algorithms move vertices to other blocks in the order of improvements in the objective. Since it allows to worsen the objective temporarily, FM can escape local optima to some extent – as opposed to simple greedy methods. However, currently only partitioners based on recursive bisection use FM-based local search algorithms [2,8,10,14,26]. On the other hand, direct k-way hypergraph partitioners [4,15,24,25] *always* employ greedy methods, although generalizations of 2-way FM to k-way partitioning have been proposed by Sanchis [21].

When improving a k-way partition directly, each vertex can potentially be moved to $k-1$ other blocks. Sanchis's algorithm maintains these moves in $k-1$ priority queues (PQs) for each block, resulting in $k(k-1)$ PQs in total. Hence, previous work on multilevel HGP notes two reasons for resorting to greedy methods: (i) Working with $k(k-1)$ PQs limits the practicality to small values of k and (ii) the Sanchis algorithm has been observed to be trapped early in local minima when used *without* the multilevel framework [12,15].

Motivation and Contribution. To our knowledge, the reasons above have kept other partitioners from evaluating direct k-way local search algorithms based on FM in the multilevel context. The present paper closes this gap with the following contributions, described in detail in Section 3: (i) We present the first direct k-way n-level hypergraph partitioner. It is motivated by the success of n-level graph partitioning [19] that performs a very fine-grained coarsening by only contracting a single edge on each level of the multilevel hierarchy. (ii) Generalizing a greedy local search method based on size-constrained label propagation (SCLaP) [18], we provide indication that greedy algorithms may work well in some cases, but cannot escape from relatively poor local optima in others. (iii) We therefore propose a localized FM-based k-way local search algorithm along the lines of Sanchis [21] that is started with a single pair of vertices only.

On 164 out of 252 experiments on well established benchmark sets, our FM-based k-way local search computes better partitions than both hMetis and PaToH and produces partitions of equal quality in 17 out of the 88 remaining cases (see Section 4). Moreover, our algorithm is about as fast as hMetis. The speed of our algorithm is mainly due to (i) a semi-dynamic hypergraph data structure, (ii) engineering the coarsening phase with the aim of uniform coarsening, and (iii) employing various speed-up techniques [7,11,16] to accelerate the gain update step, which is the main bottleneck of most FM implementations [20].

2 Preliminaries

An *undirected hypergraph* $H = (V, E, c, \omega)$ is defined as a set of vertices V and a set of hyperedges E with vertex weights $c : V \to \mathbb{R}_{\geq 0}$ and hyperedge weights $\omega : E \to \mathbb{R}_{>0}$, where each hyperedge is a subset of the vertex set V (i.e., $e \subseteq V$). We use n to denote the number of hypernodes and m to denote the number of hyperedges. In HGP literature, hyperedges are also called *nets* and the vertices of a net are called *pins* [8]. We extend c and ω to sets, i.e., $c(U) := \sum_{v \in U} c(v)$ and $\omega(F) := \sum_{e \in F} \omega(e)$. A vertex v is *incident* to a net e if $\{v\} \subseteq e$. We use $\mathrm{I}(v)$ to denote the set of all nets incident to a vertex v. The *degree* $d(v)$ of a vertex v is the number of its incident nets: $d(v) := |I(v)|$. Two vertices are *adjacent* if there exists a net e that contains both vertices. The set $\Gamma(v) := \{u \mid \exists\ e \in E : \{v, u\} \subseteq e\}$ denotes of neighbors of v. The *size* $|e|$ of a net e is the number of its pins. Nets of size one are called *single-node* nets.

A *k-way partition* of a hypergraph H is a partition of its vertex set into k *blocks* $\Pi = \{V_1, \dots, V_k\}$ such that $\bigcup_{i=1}^{k} V_i = V$, $V_i \neq \emptyset$ for $1 \leq i \leq k$ and $V_i \cap V_j = \emptyset$ for $i \neq j$. We use $b[v]$ to refer to the block id of vertex v. We call a k-way partition Π *ε-balanced* if each block $V_i \in \Pi$ satisfies a *balance constraint*: $\forall i \in \{1..k\} : |V_i| \leq L_{max} := (1 + \varepsilon)\lceil \frac{|V|}{k} \rceil$ for some parameter ε. We call a block V_i *overloaded* if $|V_i| > L_{max}$ and *underloaded* if $|V_i| < L_{max}$.

Given a k-way partition Π, the number of pins of a net e in block V_i is defined as $\Phi(e, V_i) := |\{v \in V_i \mid v \in e\}|$. If $\Phi(e, V_i) > 0$, we say that net e *is connected* to block V_i. Similarly, we say that a block V_i is *adjacent* to a vertex $v \notin V_i$ if $\exists\ e \in I(v) : \Phi(e, V_i) > 0$. $\mathrm{R}(v)$ denotes the set of all blocks adjacent to v. For each net e, $\Lambda(e) := \{V_i \mid \Phi(e, V_i) > 0\}$ denotes the *connectivity set* of e. We define the *connectivity* of a net e as the cardinality of its connectivity set: $\lambda(e) := |\Lambda(e)|$ [8]. We call a net *internal* if $\lambda(e) = 1$ and *cut* net otherwise (i.e., $\lambda(e) > 1$). Analogously a vertex that is contained in at least one cut net is called *border vertex*.

The *k-way hypergraph partitioning problem* is to find an ε-balanced k-way partition of a hypergraph H that minimizes the *total cut* $\omega(E')$ where $E' := \{e \in E : \lambda(e) > 1\}$ for some ε. This problem is known to be NP-hard [17].

Contracting a pair of vertices (u, v) means merging v into u. The weight of u becomes $c(u) := c(u) + c(v)$ and we connect u to the former neighbors $\Gamma(v)$ of v. We refer to u as the *representative* and v as the *contraction partner*. This process can lead to parallel nets (i.e., $\exists\ e_i, e_j \in E : e_i \bigtriangleup e_j \neq \emptyset$, where $\bigtriangleup$ is the symmetric difference). In this case, we choose net e_i as representative, update its weight to $\omega(e_i) = \omega(e_i) + \omega(e_j)$ and remove e_j from the hypergraph. If a contraction creates single-node nets we remove them from the hypergraph, since such nets can never become part of the cut. *Uncontracting* a vertex u undoes the contraction and restores removed parallel and single-node nets. The uncontracted vertex v is put in the same block as u and the weight of u is set back to $c(u) := c(u) - c(v)$.

3 n-Level k-way Hypergraph Partitioning

We now present our main contributions. A high-level overview of our n-level hypergraph partitioning framework is provided in Algorithm 1. As other multilevel algorithms our algorithm has a coarsening, initial partitioning and an uncoarsening phase. During the coarsening phase, we successively shrink the hypergraph by contracting only *a single pair* of vertices *at each level*, until it is small enough to be initially partitioned by some other partitioning algorithm. We desribe the details of our coarsening algorithm in Section 3.1 and briefly discuss initial partitioning in Section 3.2. The initial solution is transfered to the next finer level by performing a *single* uncontraction step. Afterwards, one of our localized local search algorithms described in Section 3.3 and Section 3.4 is used to further improve the solution quality.

Hypergraph Data Structure. Traditional multilevel algorithms create a new hypergraph for each level of the hierarchy. This is not feasible in the n-level context, since storing each level explicitly would lead to quadratic space consumption. We therefore designed a *semi-dynamic* hypergraph data structure that supports efficient contraction and uncontraction operations. Conceptually, we represent the hypergraph H as an undirected *bipartite* graph $G = (W, F)$. The vertices and nets of H form the vertex set W. For each net e incident to a vertex v, we add an edge (e, v) to the graph. The edge set F is thus defined as $F := \{(e, v) \mid e \in E : \{v\} \subseteq e\}$. When contracting a vertex pair (u, v), we mark v as deleted. The edges (v, w) incident to v are treated as follows: If G already contains an edge (u, w), then net w contained both u and v before the contraction. In this case, we simply delete the edge (v, w) from the graph. Otherwise, net w only contained v. We therefore have to relink the edge (v, w) to u. Representing this graph using an adjacency array allows us to implement deletion and relink operations with very little space overhead.

After initial partitioning, we initialize the connectivity set $\Lambda(e)$ as well as the pin counts $\Phi(e, V_i)$ for each cut net e. These data structures are then maintained and updated during the local search phase.

Algorithm 1: Multilevel Hypergraph Partitioning Framework

Input: Hypergraph H, number of desired blocks k, balance parameter ε.

while H *is not small enough* **do** *// coarsening phase*
- $(u, v) := \text{argmax}_{u \in V}\ score(u)$ *// choose vertex pair with highest rating*
- $H :=$ `contract`(H, u, v) *// $H := H \setminus \{v\}$*

$\Pi :=$ `partition`(H, k, ε) *// initial partitioning phase*

while H *is not completely uncoarsened* **do** *// uncoarsening phase*
- $(H, \Pi, u, v) :=$ `uncontract`(H, Π)
- $(H, \Pi) :=$ `refine`$(H, \Pi, u, v, k, \varepsilon)$

Output: ε-balanced k-way partition $\Pi = \{V_1, \ldots, V_k\}$

3.1 Coarsening

The vertex pairs (u, v) to be contracted are chosen according to a rating function. The goal of the coarsening phase is to contract highly connected vertices such that the number of nets remaining in the hypergraph and their size is successively reduced [14]. Removing nets leads to simpler instances for initial partitioning, while small net sizes allow FM-based local search algorithms to identify moves that improve the solution quality. Our coarsening algorithm therefore prefers vertex pairs that have a large number of heavy nets with small size in common. This score is then inversely scaled with the product of the vertex weights $c(v)$ and $c(u)$ to keep the vertex weights of the coarse hypergraph reasonably uniform:

$$r(u,v) := \frac{1}{c(v) \cdot c(u)} \sum_{e \in \{I(v) \cap I(u)\}} \frac{\omega(e)}{|e| - 1}. \tag{1}$$

This scaling factor was already effective in n-level graph partitioning [19]. At the beginning of the coarsening algorithm, all vertices are rated in random order, i.e., for each vertex u we compute the ratings of all neighbors $\Gamma(u)$ and choose the vertex v with the highest rating as contraction partner for u. Ties are broken randomly. For each vertex, we insert the vertex pair with the highest score into an addressable PQ using the rating score as key. This allows us to efficiently choose the next vertex pair that should be contracted. After contraction, we remove v from the PQ. We then remove all parallel- and single-node nets in $I(u)$. The latter are easily identified, because $|e| = 1$. For parallel hyperedge detection we use an efficient algorithm similar to the one in [13], which is used to identify vertices with identical structure in a graph: We create a fingerprint for each net $e : f_i := \bigoplus_{v \in e} v \oplus x$, for some seed x. These fingerprints are then sorted, which brings potentially parallel nets together. A final scan over the fingerprints then identifies parallel nets: Only if two consecutive fingerprints f_i, f_j are identical, we have to check whether $e_i \triangle e_j = \emptyset$ by comparing their pins.

Since each contraction potentially influences the rating scores of all neighbors $\Gamma(u)$, we have to recalculate their ratings and update the priority queue accordingly. To avoid unbalanced inputs for the initial partitioning phase, vertices v with $c(v) > c_{max} := s \cdot \frac{c(V)}{t}$ are never allowed to participate in a contraction step and are thus removed from the priority queue. The parameter s will be chosen in Section 4 and t is the maximum size of the coarsest graph, which we set to $160k$. We refer to this algorithm as *full*.

As in the graph partitioning case, the n-level approach has the advantage that it obviates the need to employ a matching or clustering algorithm to determine the vertices to be contracted. However, this comes at the expense of continuously re-rating the neighbors $\Gamma(u)$ adjacent to the representative. In hypergraph partitioning, this is the most expensive part of the algorithm, because after each contraction, we have to look at all pins of all incident nets $I(v)$. The re-rating can therefore easily become the bottleneck – especially if H contains large nets. To improve the running time of the coarsening phase in these cases, we developed two variations of the full algorithm. Both variations only differ in the way the re-rating of adjacent vertices is handled. After contracting the vertex pair (u, v), the first version only updates the rating of those neighbors, which had chosen either the representative u or the contracted vertex v as contraction partner. This can be done efficiently by maintaining the set $L_w := \{u \mid (u, w) \in \mathrm{PQ}\}$ of all representatives that choose w as contraction partner. The re-rating step then only reevaluates the rating function for each vertex in $L_u \cup L_v$. All other ratings are left untouched. We refer to this version as *partial*. The second variation does not re-rate any vertices immediately after the contraction. Instead, all adjacent vertices $\Gamma(u)$ are marked as *invalid*. If the priority queue returns an invalid vertex, we recalculate its rating and update the priority queue accordingly. In case the queue returns a valid rating, we normally continue with the coarsening process. This version is referred to as *lazy*.

3.2 Initial Partitioning

The coarsening process is repeated until the number of remaining vertices is below $160k$ or the priority queue becomes empty. The latter can happen if no valid contraction step remains, e.g., a step that would not lead to a representative u having weight $c(u) > c_{max}$. The hypergraph is then small enough to be initially partitioned by an initial partitioning algorithm. Our framework allows using hMetis or PaToH as initial partitioner. Because hMetis produces better initial partitions than PaToH, we use the recursive bisection variant of hMetis for initial partitioning. In this variant of hMetis, the maximum allowed imbalance of a partition is defined differently [14]: An imbalance value of 5, for example, allows each block to weigh between $0.45 \cdot c(V)$ and $0.55 \cdot c(V)$ *at each bisection step*. We therefore translate our maximum allowed block weight to match this definition, i.e., we use imbalance parameter

$$\varepsilon' := 100 \cdot \left(\left(\frac{1+\varepsilon}{k} + \frac{\max_{v \in V} c(v)}{c(V)} \right)^{\frac{1}{\log_2(k)}} - 0.5 \right) \tag{2}$$

for initial partitioning with hMetis. We call the initial partitioner multiple times with different random seeds and use the best partition as initial partition of the coarsest graph.

3.3 Localized direct k-way FM Local Search

Our local search algorithm follows ideas similar to the k-way FM-algorithm proposed by Sanchis [21] and is further inspired by the local search algorithm used by Sanders and Osipov [19]. Sanchis uses $k(k-1)$ priority queues to be able to maintain all possible moves for all vertices. We reduce the number of priority queues to k – one queue P_i for each block V_i. In contrast to Sanchis, we only consider to move a vertex to *adjacent* blocks rather than calculating and maintaining gains for moves to *all* blocks. This simultaneously reduces the memory requirements and restricts the search space of the algorithm to moves that are more likely to improve the solution. Another key difference is the way a local search pass is started: Instead of initializing the priority queues with all vertices or all border vertices, we perform a highly localized local search starting only with the represenative and the just uncontracted vertex. The search then gradually expands around this vertex pair by successively inserting moves for neighboring vertices into the queues.

Algorithm Outline. At the beginning of a local search pass, all queues are empty and disabled. A disabled priority queue will not be considered when searching for the next move with the highest gain. All vertices are labeled inactive and unmarked. Only unmarked vertices are allowed to become active. To start the local search phase after each uncontraction, we activate the representative and the just uncontracted vertex if they are border vertices. Otherwise, no local search phase is started. *Activating* a vertex v means that we calculate the *gain* $g_i(v)$ for moving v to all adjacent blocks $i \in R(v) \setminus \{b[v]\}$ and insert v into the corresponding queues P_i using $g_i(v)$ as key. The gain $g_i(v)$ is defined as:

$$g_i(v) := \sum_{e \in I(v)} \{\omega(e) : \Phi(e,i) = |e| - 1\} - \sum_{e \in I(v)} \{\omega(e) : \lambda(e) = 1\}. \qquad (3)$$

Thus, instead of considering all $k-1$ possible moves of a vertex v, we only examine moves to those blocks that are in the union of the connectivity sets of its incident nets: $\bigcup_{e \in I(v)} \{\Lambda(e) \setminus b[v]\}$. After insertion, all PQs corresponding to *underloaded* blocks become enabled. Since a move to an overloaded block will never be feasible, all queues corresponding to overloaded blocks are left disabled. The algorithm then repeatedly queries only the *non-empty, enabled* queues to find the move with the highest gain $g_i(v)$, breaking ties randomly. Vertex v is then moved to block V_i and labeled inactive and marked. Since each vertex is allowed to move at most once during each pass, we remove all other moves of v from the PQs. We then update all neighbors of v and continue local search until either no non-empty, enabled PQ remains or a constant number of c moves neither decreased the cut nor improved the current imbalance. The latter criterion is necessary, because otherwise the n-level approach could lead to $|V|^2$ local search steps in total. After local search is stopped, we undo all moves until we arrive at the lowest cut state reached during the search that fulfills the balance constraint. All vertices become unmarked and inactive and the algorithm is then repeated until no further improvement is achieved.

Activation and Gain Computation. Our gain computation algorithm is detailed in Algorithm 2. For each vertex v, we are only interested in gain values for moves to adjacent blocks $R(v)$. Calculating these gains can be done efficiently by looking at all incident nets $I(v)$ and all adjacent blocks *exactly once.* While iterating over $I(v)$, we generate the set $R(v)$ of all adjacent blocks (lines 11 and 16). To calculate the gain, we distinguish three cases for each net e: If $\lambda(e) = 1$ and $|e| > 1$, net e is internal in block $b[v]$ and will become a cut net when v is moved to another block. We maintain the sum of the weights of all internal nets in ω_{int}. If $\lambda(e) = 2$ and one block V_i of the two blocks in the connectivity set

Algorithm 2: Activation

1 **Function** `Activate`(v,G)
  Input: Vertex v to be activated, gain array $\Omega[1..k]$, $\forall 1 \leq i \leq k : \Omega[i] = 0$
2 **if** v is not a border vertex **then return** // *only activate border vertices*
3 $\omega_{int} := 0$ // *weight of internal nets in* $I(v)$
4 $R := \{\}$ // *discovered blocks adjacent to* v
5 **foreach** $e \in I(v)$ **do** // *visit incident nets*
6   **switch** $\lambda(e)$ **do** // *and look at connectivity*
7   **case** 1 : // *e is internal*
8   **if** $|e| > 1$ **then** $\omega_{int} := \omega_{int} + \omega(e)$
9   **case** 2 : // *e might be removable from the cut*
10   **foreach** $V_i \in \Lambda(e)$ **do** // *visit all connected blocks*
11         $R := R \cup \{V_i\}$
12   **if** $\Phi(e, V_i) = |e| - 1$ **then** // *move removes e from the cut.*
13           $\Omega[V_i] := \Omega[V_i] + \omega(e)$
14   **otherwise** // *e will not be removable from the cut*
15   **foreach** $V_i \in \Lambda(e)$ **do**
16         $R := R \cup \{V_i\}$ // *to find moves with negative or zero gain*
17 $R := R \setminus \{b[v]\}$ // *remove current block of v*
18 $\Omega[b[v]] := 0$ // *and reset the gain value*
19 **foreach** $V_i \in R$ **do** // *R now contains all adjacent blocks*
20   P_i.insert(v, $\Omega[V_i] - \omega_{int}$) // $g_i(v) = \Omega[V_i] - \omega_{int}$
21   $\Omega[V_i] := 0$ // *reset slot to initial state*
22   **if** $c(V_i) < L_{max}$ **then** P_i.enable() // *enable eligible PQs*
  Output: $\forall$ adjacent blocks V_i of v : P_i contains v with priority $g_i(v)$. If block V_i is underloaded, priority queue P_i is enabled.

$\Lambda(e)$ contains all but one pin, net e can be removed from the cut by moving v to block V_i. The weight of these nets is stored in $\Omega[V_i]$. All other nets cannot be removed from the cut by moving v to a different block. We therefore just update R accordingly. Finally, by iterating over the set of all adjacent blocks, we can compute the gain values for moving v to all connected blocks V_i by substracting the internal weight ω_{int} from the weight stored in $\Omega[V_i]$ (line 20).

Update of Neighbors. After moving a vertex v from block V_{from} to a different block V_{to}, we have to update all of its neighbors $\Gamma(v)$. All previously inactive neighbors are activated using Algorithm 2. All neighbors that became internal are labeled inactive and all corresponding moves are deleted from the priority queue. Finally, we update the gains for all moves of the remaining neighbors that are already active and remain border vertices. We reuse the gain values that are already calculated and only perform *delta-gain-updates*: If the move changed the

Algorithm 3: Delta-Gain-Update

1 **Function** `DeltaGainUpdate`($v, V_{\text{from}}, V_{\text{to}}$)
  Input: Vertex v that was moved from block V_{from} to V_{to}
2   **foreach** $e \in I(v)$ **do** *// walk all incident nets*
3   **foreach** $u \in e \setminus \{v\}$ **do** *// and consider each pin*
4   **if** $\Phi(e, V_{\text{from}}) = |e| - 1$ **then** *// move made e a cut net*
5   **foreach** $V_i \in \Pi \setminus \{V_{\text{from}}\}$ **do** P_i.`update`$(u, \omega(e))$
6   **if** $\Phi(e, V_{\text{to}}) = |e|$ **then** *// move removed e from the cut*
7   **foreach** $V_i \in \Pi \setminus \{V_{\text{to}}\}$ **do** P_i.`update`$(u, -\omega(e))$
8   **if** $\Phi(e, V_{\text{to}}) = |e| - 1 \wedge b[u] \neq V_{\text{to}}$ **then** *// only v still outside V_{to}*
9   *// moving it to V_{to} would remove e from the cut*
10   P_{to}.`update`$(u, \omega(e))$
11   **if** $\Phi(e, V_{\text{from}}) = |e| - 2 \wedge b[u] \neq V_{\text{from}}$ **then** *// 2 pins outside V_{from}*
12   *// moving v to V_{from} could have removed e from the cut*
13   P_{from}.`update`$(u, -\omega(e))$
  Output: The gains for all moves of all neighbors $\Gamma(v)$ are updated.

contribution to the gain values for a net $e \in I(v)$, we account for that change by incrementing/decrementing the gains of the corresponding moves by $\omega(e)$. Our delta-gain-update algorithm is outlined in Algorithm 3. For each net e, we have to consider four cases:

1. Before the move, net e was completely internal in block V_{from}. Now, after the move, e has become a cut net with $\lambda(e) = 2$ and $\Lambda(e) = \{V_{\text{from}}, V_{\text{to}}\}$, if all but one pin of net e are in block V_{from}. In this case, before the move, net e contributed $-\omega(e)$ to the gain of all its pins for moving to another block. Since the move of vertex v now made e a cut net, all other pins can be moved to another block without incurring a further increase in cut. We therefore change the contribution of net e from $-\omega(e)$ to zero by increasing the corresponding gains by $\omega(e)$ (lines 4 to 5).
2. Vertex v was the only pin of net e that was outside of block V_{to} before the move and the movement therefore removed e from the cut. Now that e is internal, it contributes $-\omega(e)$ to the gain of all its remaining pins for moving to another part, since each move would again make it a cut net (lines 6 to 7).
3. After the move of v only one pin of net e remains outside of V_{to}. If that pin is also moved to V_{to}, we remove e from the cut. The corresponding move of this pin therefore receives a delta-gain-update of $\omega(e)$. For all other pins, the contribution of e to their gain values did not change (lines 8 to 10).
4. Before the move of v, there was only one pin left that was outside of V_{from}. Moving this pin to V_{from} would have removed e from the cut. However, now that v is also outside of V_{from}, the move of this pin cannot decrease the cut

any more. The contribution of net e to the gain of moving this pin to V_{from} therefore changes from $\omega(e)$ to zero (lines 11 to 13).

For each active vertex, the priority queues maintain the gains to all *adjacent* blocks. The set of adjacent blocks, however, is subject to change during local search, because vertex movements can increase as well as decrease the connectivity of incident nets. The update process therefore has to take these changes into account. Otherwise we would either miss potential moves or perform *stale* moves, i.e., move a vertex to a block that was adjacent to v at some point of the local search but is not adjacent any more at the time it is returned by the priority queue. The movement of v increased the set of adjacent blocks $\mathrm{R}(u)$ for one of its neighbors u if $V_{\mathrm{to}} \notin \mathrm{R}(u)$ before the move. In this case, we calculate the gain $g_{\mathrm{to}}(u)$ and insert u into the priority queue P_{to}. Similarly, if the movement decreased the set of adjacent blocks (i.e., $V_{\mathrm{from}} \notin \mathrm{R}(u)$ after the move), we remove the vertex from P_{from}.

Critical Nets. Having identified the cases in which a move changes the gain contribution of one of its incident nets, we generalize the notion of *critical nets* introduced by Fiduccia and Mattheyses [11] for bipartitioning to k-way partitioning. A net is said to be *critical* if there exists at least one move for one of its pins that affects the gain contribution. Notice that the conditions concerning $\Phi(e, \cdot)$ in lines 4, 6, 8 and 11 of Algorithm 3 can be evaluated without considering a pin of the corresponding net. Thus, we can determine whether or not a net is critical by evaluating these conditions once before iterating over all pins (line 3). Only if one of the conditions evaluates to true, we actually have to consider each pin of the net.

Locked Nets. Performing delta-gain-updates rather than re-calculating the gains for each neighbor of a moved vertex from scratch considerably reduces the complexity of the update step. The complexity can be reduced even further by noticing that the contribution of a net does not change any more once two of its pins have been moved to two *different* blocks. The net is then *locked* in those two blocks, because neither of the two vertices is allowed to be moved again during the current local search pass. It is therefore not possible to remove such a net from the cut by moving any of the remaining movable pins to another block. Thus it is not necessary to perform any further delta-gain-updates for locked nets. This observation was first described by Krishnamurthy [16] for bipartitioning and transferred to k-way partitioning by Sanchis [21]. We integrate locking of nets into our algorithm by labeling each net during a local search pass. Initially, all nets are labeled *free.* Once the first pin of a net is moved, the net becomes *loose.* It now has a pin in one block that cannot be moved again. Further moves to this block do not change the label of the net. As soon as another pin is moved to a different block, the net is labeled *locked* and is excluded from future delta-gain-updates. However, we still have to account for changes in connectivity as described above.

3.4 Local Search with Size-Constrained Label Propagation (SCLaP)

The *label propagation algorithm* for graph clustering was recently equipped with a size constraint to work as a coarsening and a local search algorithm for graph partitioning [18]. We briefly outline the previous local search algorithm before describing our adaptation to hypergraphs. Initially, each vertex v is in its current block $b[v]$. The algorithm then works in rounds. In each round, the vertices are visited in random order and each vertex v is moved to the eligible (i. e. not overloaded after the move) block V_i that has the strongest connection to v. Ties are broken randomly. After all vertices are visited, the process is repeated until the labels have converged or a maximum number of ℓ rounds is reached.

We modify this local search algorithm as follows in order to be applicable in our n-level hypergraph partitioning context (see Algorithm 4). Instead of iterating over all vertices, we start the first iteration only with the vertex pair (u, v) that has just been uncontracted. If one of these two vertices changes its block, all of its neighbors are allowed to change their block in the next iteration. This can be done efficiently by maintaining two queues Q_1 and Q_2. Q_1 contains the vertices for the current iteration and Q_2 those for the next iteration. After each round, we clear Q_1 and swap it with Q_2. In order to reflect our partitioning objective, we move a vertex to the eligible block that maximizes the *gain* as

Algorithm 4: Label Propagation Local Search

Input: Uncontracted vertex pair (u, v)

$Q_1 := \{u, v\}$ *// set of vertices for current iteration*
$Q_2 := \{\}$ *// set of vertices for next iteration*
while $Q_1 \neq \{\} \wedge num\_iterations \leq max\_iterations$ **do**
  foreach $v \in Q_1$ in random order **do**
    $\Omega[1..k] := \perp$
    foreach $V_i \in R(v)$ **do** *// for all adjacent blocks*
      foreach $e \in I(v)$ **do** *// calculate gains for moves*
        if $c(v) + c(V_i) \leq L_{max}$ **then** *// enforce balance constraint*
          if $\Omega[V_i] = \perp$ **then** $\Omega[V_i] := 0$
          $\Omega[V_i] := \Omega[V_i] + \mathtt{gain}(v, e, V_i)$
    $V_{max} := \mathrm{argmax}_{V_i}\, \Omega[V_i]$ *// choose max-gain move with max. Λ-decrease*
    if $b[v] \neq V_{max}$ **then**
      $\mathtt{move}(v, V_{max})$
      foreach $w \in \Gamma(v)$ **do**
        // all neighbors may change their block in next round
        $Q_2 := Q_2 \cup \{w\}$
  $Q_1 := \{\}$
  $\mathtt{swap}(Q_1, Q_2)$

Output: Refined partition $\Pi = \{V_1, \ldots, V_k\}$

defined in Equation 3. For each incident net e, we calculate its contribution to the gain for moving it to each adjacent block V_i (line 6 – line 10):

$$\texttt{gain}(v, e, V_i) := \begin{cases} -\omega(e) & \text{if } \lambda(e) = 1 \land V_i \notin \Lambda(e) \\ \omega(e) & \text{if } \lambda(e) = 2 \land \Phi(e, V_i) = |e| - 1 \\ 0 & \text{else} \end{cases} \quad (4)$$

Finally, we adapt the tie-breaking scheme: If multiple blocks have the same maximum gain, we choose to move the vertex to the block that leads to the highest connectivity decrease for all incident nets. A move of vertex v to block V_i decreases the connectivity of a net e, if $\Phi(e, b[v]) = 1$ and $\Phi(e, V_i) \neq 0$. Analogously, a move increases the connectivity if $\Phi(e, V_i) = 0$. The total connectivity decrease for moving a vertex v to block V_i can therefore be computed during the gain calculation. The intention behind this tie-breaking scheme is to successively reduce the number of blocks a net is connected to. Only in case multiple blocks also have the same connectivity decrease value, we resort to random tie breaking.

3.5 Iterated Multilevel Algorithms

V-cycles are a common technique to further improve a solution [14,22,27]. The idea is to reuse an already computed partition as input for the multilevel approach. During coarsening, the quality of the solution is maintained by only contracting vertices belonging to the same block. The current partition of the coarsest graph is then used as initial partition . During uncoarsening, local search algorithms can then further improve solution quality. We also adopt this technique for n-level hypergraph partitioning by modifying the rating algorithm such that we only allow the contraction of vertex pairs that belong to the same block.

4 Experiments

Instances. We evaluate our algorithms on hypergraphs derived from two well established benchmark sets: The ISPD98 Circuit Benchmark Suite [1] and the University of Florida Sparse Matrix Collection [9]. From the latter, we use the instances that are part of the 10th DIMACS implementation challenge dataset [5]. The matrices are translated into hypergraphs using the row-net model, i.e. each row of the matrix is treated as a net. All hypergraph have unit net and vertex weights. We exclude the two largest instances *nlpkkt200* and *nlpkkt240*, because they could not be partitioned using hMetis. In both cases the amount of memory needed by hMetis exceeded the amount of memory available on our machine.

We divided the benchmark set into *medium-sized* and *large* instances and use $k \in \{2, 4, 8, 16, 32, 64, 128\}$ for the number of blocks and an allowed imbalance of $\varepsilon = 0.03$. The properties of the hypergraphs are summarized in Table 5.

System. All experiments are performed on a single core of a machine consisting of two Intel Xeon E5-2670 Octa-Core processors (Sandy Bridge) clocked at 2.6 GHz. The machine has 64 GB main memory, 20 MB L3-Cache and 8x256 KB L2-Cache and is running Ret Hat Enterprise Linux (RHEL) 6.4.

Methodology. The algorithms are implemented in the n-level hypergraph partitioning framework *KaHyPar* (**Ka**rlsruhe **Hy**pergraph **Par**titioning). The code is written in C++ and compiled using gcc-4.9.1 with flags `-O3 -mtune=native -march=native`. Unless otherwise mentioned, we perform ten repetitions with different seeds for each experiment and report the arithmetic mean of the computed cut and running time as well as the best cut found. When averaging over different instances, we use the geometric mean in order to give every instance a comparable influence on the final result. We compare our algorithms to both the k-way (hMetis-K) and the recursive-bisection variant (hMetis-R) of hMetis 2.0 (p1) [14,15] and to PaToH 3.2 [8]. As noted in Section 3.2, hMetis-R employs a different balance constraint. We therefore translate our imbalance parameter $\varepsilon = 0.03$ to ε' as described in Equation 2 such that it matches our balance constraint after $\log_2(k)$ bisections. Since PaToH ignores the random seed if configured to use the quality preset, we report both the result of the quality preset (PaToH-Q) as well as the average over ten repetitions using PaToH in default configuration (PaToH-D). In both cases, we configured PaToH to use a final imbalance ratio of $\varepsilon = 0.03$ to match our balance constraint.

Algorithm Configuration. We performed a large amount of experiments to tune the parameters of our algorithms using the medium-sized instances and $k \in \{2, 4, 8, 16, 32, 64\}$. A full description of these experiments is omitted due to space constraints. We use two sets of parameter settings *fast* and *strong* that turned out to work well. Both configurations use the same parameters for coarsening and initial partitioning: We use the *lazy* variant as coarsening algorithm, because it is the fastest algorithm and provides comparable quality to both other variants (see Table 1). The coarsening process is stopped as soon as the number of vertices drops below $t = 160k$ or no eligible vertex is left. The scaling factor s for the maximum allowed vertex weight during coarsening is set to 2.5. We use the default configuration of hMetis as initial partitioner and perform only one initial partitioning trial, since more iterations seldomly produced different cuts. The strong configuration uses the k-way FM algorithm described in Section 3.3 and stops each pass as soon as $c = 200$ moves did not yield any improvement. The fast configuration employs the SCLaP-based refinement algorithm presented in Section 3.4 and performs $\ell = 5$ rounds on each level. Both configurations can be augmented using V-cycles (referred to as *FastV* and *StrongV*). The maximum number of V-cycle iterations is set to 3 for FastV and to 10 for StrongV.

Variant	avg. cut	best cut	$\frac{\text{coarsening time}}{\text{coarsening time}_{full}}$
full	2 990.95	2 910.30	1.00
partial	2 996.93	2 914.21	0.10
lazy	2 988.74	2 916.62	0.06

Table 1. Test results for the three different variants of our coarsening algorithm on medium-sized instances. Running time of the coarsening phase is relative to *full*.

4.1 Main Results

Evaluation of Local Search Algorithms. In the following, we report the final results of the parameter tuning on the medium-sized benchmark set and evaluate the performance of our local search algorithms described in Section 3.3 and Section 3.4 in the n-level context. The results are summarized in Table 2. Using an FM-based heuristic pays off: The localized k-way FM algorithm consistently outperforms the greedy SCLaP-based algorithm. The cuts of label propagation using V-cycles are on average 4% larger than those of StrongV. Its advantage decreases for larger values of k. This can be explained by the fact that as k increases, cut nets are more likely to connect more than two blocks. The FM algorithm then has to deal with an increasing number of zero-gain moves, which effectively weakens its ability to climb out of local minima.

V-cycles improve the solution quality of both algorithms by around 1% on average. The impact of global search iterations is larger for the SCLaP-based algorithm than for localized k-way local search, especially if the number of blocks is small. In these cases, it is more likely that a vertex can switch its label and thereby improve the solution quality. With increasing k, this becomes more difficult. The effect of V-Cycles is more stable for k-way FM algorithm, since it is already a strong heuristic.

Considering running times, we note that the SCLaP-based algorithm is an order of magnitude faster on average than the localized FM algorithm. However, best cuts found by the former are still larger than the average cuts of the latter – even for instances of medium size.

Table 2. Performance of our local search algorithms on the medium-sized instances used for parameter tuning. All average cuts and best cuts are shown as increases (%) relative to the values obtained by StrongV.

k	StrongV			Strong			FastV			Fast		
	best	avg	t[s]	best[%]	avg[%]	t[s]	best[%]	avg[%]	t[s]	best[%]	avg[%]	t[s]
2	792	815	13.7	+0.56	+0.52	5.6	+4.50	+5.70	1.3	+6.15	+8.39	0.7
4	1 662	1 744	37.4	+1.54	+1.02	11.6	+5.17	+5.31	1.9	+7.95	+7.77	1.1
8	2 823	2 920	76.3	+1.66	+1.18	20.3	+5.43	+5.52	2.8	+7.58	+7.68	2.0
16	4 090	4 190	155.7	+1.54	+1.45	33.5	+4.57	+4.50	4.6	+5.95	+5.94	3.7
32	5 454	5 529	229.1	+1.17	+1.06	45.3	+3.32	+3.18	7.8	+4.19	+3.98	7.0
64	6 843	6 921	228.8	+0.92	+0.85	51.2	+2.42	+2.32	14.1	+2.84	+2.76	13.3
avg	2 877	2 955	82.7	+1.23	+1.01	21.6	+4.23	+4.41	3.9	+5.76	+6.07	2.8

Comparison to other Hypergraph Partitioners. We now switch to our benchmark set containing large instances to avoid the effect of overtuning our algorithms to the instances used for parameter tuning. We exclude *cage15* from the following results, because hMetis took more than 18 hours to compute a single bipartition. hMetis-K is the only algorithm that often produces imbalanced partitions. Out of 1610 cases, 209 partitions are imbalanced (up to 12% imbalance). It therefore

Table 3. Comparison of our algorithms and state-of-the-art hypergraph partitioners on large benchmark instances.

Algorithm	avg. cut	best cut	t[s]
StrongV	12 968	12 706	979.0
Strong	13 088	12 815	211.3
FastV	13 955	13 581	61.5
Fast	14 269	13 861	30.7
hMetis-R	13 155	12 977	230.7
hMetis-K	13 548	13 341	134.3
PaToH-Q	13 805	13 805	12.6
PaToH-D	14 560	13 912	3.1

has slight advantages in the following comparisons because we do not disqualify imbalanced partitions. Table 3 and Table 4 summarize the results. Detailed per-instance results can be found in the Appendix in Table 6.

The strong variants of KaHyPar produce the smallest average and minimum cuts. The results of KaHyPar-FastV are comparable to the cuts produced by PaToH. On average, the cuts produces by PaToH-D, PaToH-Q, hMetis-K, hMetis-R are 12%, 7%, 5% and 2% larger than those of KaHyPar-StrongV, respectively. hMetis-R performs surprisingly well, considering the fact that we had to tighten the balancing constraint in order to ensure balanced solutions. However, out of 161 instances, KaHyPar-StrongV computed 112 partitions that were better than those produced by hMetis-R and reproduced the cuts of hMetis-R in 14 of the remaining 49 cases. Also note that KaHyPar-Strong dominates the previously best solver hMetis-R with respect to both quality and running time.

As can be seen in Table 4, the greedy label propagation algorithm outperforms PaToH on VLSI instances, while still being on average around four times faster than hMetis. On sparse matrix instances however, it is not able to escape from local minima and thus cannot improve the quality above PaToH's level.

Looking at the more detailed comparison of KaHyPar-StrongV to the other partitioning packages in Table 4, we see that the localized k-way FM algorithm is only beaten by hMetis-R for $k = 2$ and $k = 4$. The improvement in solution quality increases with an increasing number of blocks. For $k = 128$ our algorithm produces 7% better cuts than hMetis-K and 3% better cuts than hMetis-R. For large values of k, it becomes increasingly difficult for the greedy refinement algorithm used in hMetis-K to find moves with a positive gain. The same problem holds true for our SCLaP-based algorithm, which actually performs worse than its counterpart in hMetis-K. This could be explained by the fact that our algorithm only tries to optimize around the just uncontracted vertex pair and is likely to be trapped in a local minimum, while the greedy refinement of hMetis-K in each iteration visits all vertices and moves them to an eligible block if the move has a positive gain.

Table 4. Detailed comparison of our algorithms and other partitioners on large benchmark instances. The first table summarizes the performance of the k-way local search algorithm on *all* large instances. The second and third table show the results for the greedy algorithm on large VLSI (second) and large sparse matrix instances (third). The average cuts are shown as increases in cut (%) relative to the values obtained by our algorithm shown in the first column.

k	**StrongV**		hMetis-K		hMetis-R		PaToH-Q		PaToH-D	
	avg. cut	t[s]	cut [%]	t[s]	cut [%]	t[s]	cut [%]	t[s]	cut [%]	t[s]
2	2 563.6	141.0	+2.02	76.0	−1.84	82.0	+2.94	4.0	+8.97	1.1
4	5 471.3	313.9	+3.02	89.7	−0.07	151.4	+8.06	7.7	+13.60	2.0
8	9 382.7	684.0	+3.19	103.6	+1.22	211.7	+7.08	11.3	+14.01	2.8
16	14 760.4	1 100.0	+4.56	124.7	+2.27	267.5	+6.98	14.7	+13.46	3.5
32	21 980.8	1 884.3	+5.25	158.5	+2.44	319.9	+6.85	18.4	+12.01	4.3
64	32 190.5	3 115.7	+6.36	208.7	+2.75	369.6	+6.71	21.4	+12.50	5.0
128	44 865.1	4 407.8	+7.04	271.1	+3.46	418.8	+6.63	25.1	+11.46	5.7
avg	12 967.7	979.0	+4.48	134.3	+1.45	230.7	+6.45	12.6	+12.28	3.1

k	**FastV**		hMetis-K		hMetis-R		PaToH-Q		PaToH-D	
	avg. cut	t[s]	cut [%]	t[s]	cut [%]	t[s]	cut [%]	t[s]	cut [%]	t[s]
2	1 578.0	4.1	−4.27	15.2	−5.64	16.8	−0.16	0.9	+9.33	0.2
4	3 349.2	5.0	−6.82	19.0	−6.16	30.9	+2.25	1.7	+12.09	0.4
8	5 215.2	6.8	−5.92	23.5	−4.37	42.9	+3.25	2.5	+11.77	0.5
16	7 655.8	10.3	−4.26	31.6	−2.60	54.8	+2.61	3.2	+9.62	0.6
32	10 649.4	16.7	−2.69	46.8	−1.73	66.4	+2.78	4.1	+8.27	0.8
64	14 322.2	28.0	−0.76	72.9	−0.44	77.2	+3.57	4.8	+8.29	0.9
128	18 316.5	42.5	+2.48	106.6	+0.72	88.9	+3.21	5.6	+7.22	1.0
avg	6 673.5	11.6	−3.22	36.0	−2.92	47.6	+2.49	2.8	+9.50	0.6

k	**FastV**		hMetis-K		hMetis-R		PaToH-Q		PaToH-D	
	avg. cut	t[s]	cut [%]	t[s]	cut [%]	t[s]	cut [%]	t[s]	cut [%]	t[s]
2	4 573.2	225.8	−5.42	331.7	−10.98	350.5	−7.42	15.4	−4.98	5.0
4	10 012.8	233.9	−3.21	371.5	−9.29	650.1	−2.61	30.0	−1.45	9.2
8	18 887.7	247.0	−4.41	403.1	−9.26	914.2	−5.78	44.3	−1.19	13.1
16	31 280.3	269.6	−2.37	438.5	−7.89	1 143.5	−4.26	58.3	+0.86	16.7
32	49 073.0	300.6	−1.56	484.7	−7.39	1 352.9	−3.62	72.7	+0.57	20.3
64	77 031.8	343.0	−0.50	547.7	−7.14	1 552.8	−3.71	83.5	+2.28	23.8
128	114 556.2	398.6	−0.83	638.0	−5.59	1 733.5	−2.18	99.1	+2.83	27.3
avg	27 440.7	282.8	−2.63	449.2	−8.23	979.9	−4.24	49.4	−0.19	14.5

5 Conclusions and Future Work

We presented the n-level direct k-way hypergraph partitioning framework KaHyPar. Using a highly localized version of k-way FM, our algorithm produces better partitions than hMetis and PaToH on 73% of the VLSI instances and 71% of the sparse matrix instances. Our greedy algorithm based on size-constrained label propagation gives better quality than PaToH on VLSI instances while still being several times faster than hMetis.

Motivated by the effectiveness of hMetis-R, evaluating recursive bisection in the context of n-level partitioning seems to be a promising area of future research. Having both a direct k-way and a recursive bisection n-level partitioner, KaHyPar could then be embedded into an evolutionary framework which combines both approaches to find better solutions [23].

Throughout local search, a lot of moves have gain zero. Integrating the concept of higher-level gains as described by [16,21] therefore would be a promising approach to give these moves more meaning. The running time of our k-way local search could be improved by developing an adaptive stopping rule as in [19] that is able to model zero-gain moves and stops local search if further improvement becomes unlikely. Having shown that our localized direct k-way local search algorithm is able to optimize the total cut size, future work could also look at different partitioning objectives that rely on a global view of the problem, like the $(\lambda - 1)$ or *sum-of-external-degrees* metric [15].

References

1. C. J. Alpert. The ISPD98 Circuit Benchmark Suite. In *Proc. of the 1998 Int. Symp. on Physical Design*, ISPD '98, pages 80–85, New York, 1998. ACM.
2. C. J. Alpert, J.-H. Huang, and A. B. Kahng. Multilevel Circuit Partitioning. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 17(8):655–667, 1998.
3. C. J. Alpert and A. B. Kahng. Recent Directions in Netlist Partitioning: a Survey. *Integration, the VLSI Journal*, 19(1–2):1 – 81, 1995.
4. C. Aykanat, B. B. Cambazoglu, and B. Uçar. Multi-level Direct K-way Hypergraph Partitioning with Multiple Constraints and Fixed Vertices. *J. Parallel Distrib. Comput.*, 68(5):609–625, 2008.
5. D. Bader, A. Kappes, H. Meyerhenke, P. Sanders, C. Schulz, and D. Wagner. Benchmarking for Graph Clustering and Partitioning. In *Encyclopedia of Social Network Analysis and Mining*. Springer, 2014.
6. D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, editors. *Proc. Graph Partitioning and Graph Clustering - 10th DIMACS Implementation Challenge Workshop*, volume 588 of *Contemporary Mathematics*. AMS, 2013.
7. A. E. Caldwell, A. B.. Kahng, and I. L. Markov. Design and Implementation of the Fiduccia-Mattheyses Heuristic for VLSI Netlist Partitioning. In *ALENEX'99*, volume 1619 of *LNCS*, pages 182–198. Springer, 1999.
8. U. V. Catalyurek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. on Parallel and Distributed Systems*, 10(7):673–693, Jul 1999.

9. T. A. Davis and Y. Hu. The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, 2011.
10. K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and U. V. Catalyurek. Parallel Hypergraph Partitioning for Scientific Computing. IEEE, 2006.
11. C. Fiduccia and R. Mattheyses. A Linear Time Heuristic for Improving Network Partitions. In *19th ACM/IEEE Design Automation Conf.*, pages 175–181, 1982.
12. J. Gong and S. K. Lim. Multiway Partitioning with Pairwise Movement. In *IEEE/ACM Int. Conf. on Computer-Aided Design*, pages 512–516, 1998.
13. B. Hendrickson and E. Rothberg. Improving the Run Time and Quality of Nested Dissection Ordering. *SIAM J. on Scientific Computing*, 20(2):468–489, 1998.
14. G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel Hypergraph Partitioning: Applications in VLSI Domain. *IEEE Trans. on Very Large Scale Integration VLSI Systems*, 7(1):69–79, 1999.
15. G. Karypis and V. Kumar. Multilevel K-way Hypergraph Partitioning. In *Proc. 36th ACM/IEEE Design Automation Conference*, pages 343–348. ACM, 1999.
16. B. Krishnamurthy. An Improved Min-Cut Algonthm for Partitioning VLSI Networks. *IEEE Transactions on Computers*, C-33(5):438–446, 1984.
17. T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* John Wiley & Sons, Inc., 1990.
18. H. Meyerhenke, P. Sanders, and C. Schulz. Partitioning Complex Networks via Size-Constrained Clustering. In *Experimental Algorithms*, volume 8504 of *LNCS*, pages 351–363. Springer, 2014.
19. V. Osipov and P. Sanders. n-Level Graph Partitioning. In *Algorithms ESA 2010*, volume 6346 of *LNCS*, pages 278–289. Springer, 2010.
20. D. A. Papa and I. L. Markov. chapter Hypergraph Partitioning and Clustering, pages 61–1–61–19. Chapman and Hall/CRC, 2007.
21. L. A. Sanchis. Multiple-way Network Partitioning. *IEEE Trans. on Computers*, 38(1):62–81, 1989.
22. P. Sanders and C. Schulz. Engineering Multilevel Graph Partitioning Algorithms. In *Algorithms ESA 2011*, volume 6942 of *LNCS*, pages 469–480. Springer, 2011.
23. P. Sanders and C. Schulz. Distributed Evolutionary Graph Partitioning. In *Meeting on Algorithm Engineering & Experiments (ALENEX'12)*. SIAM, 2012.
24. A. Trifunovi and W. J. Knottenbelt. Parallel Multilevel Algorithms for Hypergraph Partitioning. *J. of Parallel and Distributed Computing*, 68(5):563 – 581, 2008.
25. U. V. Çatalyürek and M. Deveci and K. Kaya and B. Uçar. UMPa: A multi-objective, multi-level partitioner for communication minimization. In Bader et al. [6], pages 53–66.
26. B. Vastenhouw and R. H. Bisseling. A Two-Dimensional Data Distribution Method for Parallel Sparse Matrix-Vector Multiplication. *SIAM Rev.*, 47(1):67–95, 2005.
27. C. Walshaw. Multilevel Refinement for Combinatorial Optimisation Problems. *Annals of Operations Research*, 131(1-4):325–372, 2004.

A Benchmark Instances

Table 5. Properties of our benchmark set containing medium-sized instances (top) and large instances (bottom). Tables are split into two groups: VLSI instances and sparse matrix instances. Within each group, the hypergraphs are sorted by size.

Hypergraph	$\lvert V \rvert$	$\lvert E \rvert$	$\lvert pins \rvert$	$d(v)$			$\lvert e \rvert$		
				min	avg	max	min	avg	max
ibm01	12 752	14 111	50 566	1	3.97	39	2	3.58	42
ibm02	19 601	19 584	81 199	1	4.14	69	2	4.15	134
ibm03	23 136	27 401	93 573	1	4.04	100	2	3.41	55
ibm04	27 507	31 970	105 859	1	3.85	526	2	3.31	46
ibm05	29 347	28 446	126 308	1	4.30	9	2	4.44	17
ibm06	32 498	34 826	128 182	1	3.94	91	2	3.68	35
ibm07	45 926	48 117	175 639	1	3.82	98	2	3.65	25
vibrobox	12 328	12 328	342 828	9	27.81	121	9	27.81	121
bcsstk29	13 992	13 992	619 488	5	44.27	71	5	44.27	71
memplus	17 758	17 758	126 150	2	7.10	574	2	7.10	574
bcsstk30	28 924	28 924	2 043 492	4	70.65	219	4	70.65	219
bcsstk31	35 588	35 588	1 181 416	2	33.20	189	2	33.20	189
bcsstk32	44 609	44 609	2 014 701	2	45.16	216	2	45.16	216

Hypergraph	$\lvert V \rvert$	$\lvert E \rvert$	$\lvert pins \rvert$	$d(v)$			$\lvert e \rvert$		
				min	avg	max	min	avg	max
ibm08	51 309	50 513	204 890	1	3.99	1165	2	4.06	75
ibm09	53 395	60 902	222 088	1	4.16	173	2	3.65	39
ibm10	69 429	75 196	297 567	1	4.29	137	2	3.96	41
ibm11	70 558	81 454	280 786	1	3.98	174	2	3.45	24
ibm12	71 076	77 240	317 760	1	4.47	473	2	4.11	28
ibm13	84 199	99 666	357 075	1	4.24	180	2	3.58	24
ibm14	147 605	152 772	546 816	1	3.70	270	2	3.58	33
ibm15	161 570	186 608	715 823	1	4.43	306	2	3.84	36
ibm16	183 484	190 048	778 823	1	4.24	177	2	4.10	40
ibm17	185 495	189 581	860 036	1	4.64	81	2	4.54	36
ibm18	210 613	201 920	819 697	1	3.89	97	2	4.06	66
finan512	74 752	74 752	596 992	3	7.99	55	3	7.99	55
af_shell9	504 855	504 855	17 588 875	20	34.84	40	20	34.84	40
audikw_1	943 695	943 695	77 651 847	21	82.28	345	21	82.28	345
ldoor	952 203	952 203	46 522 475	28	48.86	77	28	48.86	77
ecology2	999 999	999 999	4 995 991	3	5.00	5	3	5.00	5
ecology1	1 000 000	1 000 000	4 996 000	3	5.00	5	3	5.00	5
thermal2	1 228 045	1 228 045	8 580 313	1	6.99	11	1	6.99	11
af_shell10	1 508 065	1 508 065	52 672 325	15	34.93	35	15	34.93	35
G3_circuit	1 585 478	1 585 478	7 660 826	2	4.83	6	2	4.83	6
kkt_power	2 063 494	2 063 494	14 612 663	1	7.08	96	1	7.08	96
nlpkkt120	3 542 400	3 542 400	96 845 792	5	27.34	28	5	27.34	28
cage15	5 154 859	5 154 859	99 199 551	3	19.24	47	3	19.24	47
nlpkkt160	8 345 600	8 345 600	229 518 112	5	27.50	28	5	27.50	28

B Detailed Results

Table 6: Detailed results per instance. The best cut is highlighted.

		KaHyPar-Strong			KaHyPar-StrongV			KaHyPar-Fast			KaHyPar-FastV			hMetis-K			hMetis-R			PaToH-Q			PaToH-D		
H	k	Best	Avg.	t[s]	Best	Avg.	t[s]	Best	Avg.	t[s]	Best	Avg.	t[s]	Best	Avg.	t[s]	Best	Avg.	t[s]	Best	Avg.	t[s]	Best	Avg.	t[s]
ibm01	2	**203**	243.3	1.54	**203**	241.3	3.41	266	274.0	0.22	261	266.0	0.38	**203**	206.3	0.86	**203**	203.1	1.17	252	252	0.15	265	290.3	0.03
	4	583	600.0	3.89	579	596.8	9.42	600	622.1	0.48	590	610.5	0.63	**495**	520.4	1.48	535	537.2	2.50	640	640	0.23	546	656.5	0.04
	8	864	882.9	6.47	860	875.8	18.72	907	929.5	1.04	889	908.7	1.18	809	820.2	2.52	**808**	823.4	3.80	875	875	0.34	920	978.2	0.05
	16	1 243	1 261.6	9.13	**1 230**	1 248.6	33.14	1 299	1 323.3	2.13	1 276	1 301.6	2.25	1 267	1 275.7	4.97	1 273	1 291.8	5.23	1 348	1 348	0.42	1 392	1 443.5	0.07
	32	1 657	1 687.1	10.45	**1 638**	1 658.7	45.76	1 714	1 744.3	4.18	1 704	1 729.0	4.30	1 733	1 752.4	9.02	1 715	1 732.1	6.95	1 803	1 803	0.54	1 814	1 893.6	0.09
	64	2 223	2 239.4	10.46	**2 197**	2 211.1	28.02	2 250	2 274.9	7.92	2 250	2 268.3	8.01	2 359	2 375.1	14.77	2 289	2 295.0	8.90	2 388	2 388	0.60	2 412	2 455.2	0.11
	128	2 959	2 973.1	11.23	2 959	2 973.1	11.19	2 959	2 973.1	11.17	2 959	2 973.1	11.23	3 100	3 113.6	20.21	**2 955**	2 972.3	11.42	2 973	2 973	0.74	3 087	3 113.9	0.13
ibm02	2	354	365.9	4.41	346	362.0	11.25	383	408.4	0.46	364	385.5	0.84	351	359.7	2.82	**344**	349.4	3.97	375	375	0.21	369	401.5	0.04
	4	707	721.9	8.12	696	714.7	25.35	756	805.2	0.98	726	773.3	1.35	**648**	681.8	4.01	703	714.7	6.93	705	705	0.40	689	839.2	0.08
	8	2 016	2 056.0	15.58	1 991	2 015.1	77.86	2 236	2 310.5	2.36	2 160	2 217.1	2.73	2 013	2 069.9	7.78	2 005	2 054.3	10.37	**1 963**	1 963	0.69	1 986	2 162.5	0.11
	16	3 366	3 406.5	26.89	**3 306**	3 349.6	145.98	3 581	3 616.9	5.16	3 528	3 563.1	5.49	3 410	3 448.6	14.26	3 452	3 470.4	13.31	3 398	3 398	0.84	3 448	3 549.2	0.14
	32	4 370	4 406.4	37.88	**4 296**	4 331.7	254.89	4 520	4 544.0	9.03	4 474	4 506.5	9.29	4 650	4 760.9	22.34	4 462	4 498.7	15.51	4 469	4 469	1.04	4 582	4 664.0	0.16
	64	5 189	5 218.7	40.55	**5 132**	5 171.9	248.02	5 234	5 281.3	14.00	5 220	5 265.5	14.22	5 879	5 911.5	35.45	5 310	5 337.6	17.61	5 344	5 344	1.13	5 364	5 449.7	0.19
	128	6 087	6 113.2	20.27	6 087	6 113.2	20.25	6 087	6 113.2	20.22	6 087	6 113.2	20.27	6 771	6 788.2	42.91	6 065	6 111.4	20.17	**6 027**	6 027	1.24	6 122	6 173.4	0.21
ibm03	2	959	968.4	7.63	**954**	961.7	20.96	1 006	1 049.3	0.49	991	1 015.8	0.86	959	962.4	2.60	957	960.2	2.93	989	989	0.25	990	1 016.5	0.04
	4	1 708	1 782.8	14.59	1 702	1 760.6	47.68	1 804	1 925.3	0.98	1 748	1 853.5	1.35	**1 670**	1 697.6	3.51	1 703	1 733.5	5.27	1 887	1 887	0.47	1 832	1 991.6	0.07
	8	2 494	2 625.4	22.55	2 473	2 575.7	102.70	2 623	2 817.8	1.93	2 531	2 717.0	2.27	**2 427**	2 447.9	5.68	2 504	2 521.6	7.63	2 704	2 704	0.61	2 869	3 012.9	0.10
	16	3 317	3 382.8	28.46	3 278	3 337.4	156.66	3 431	3 513.4	3.56	3 369	3 451.5	3.88	3 291	3 317.9	9.69	**3 246**	3 298.4	9.85	3 584	3 584	0.70	3 662	3 756.2	0.13
	32	4 029	4 050.4	32.44	**3 988**	4 016.9	178.72	4 136	4 182.5	6.21	4 115	4 144.4	6.48	4 194	4 213.3	15.78	4 080	4 134.8	12.15	4 254	4 254	0.97	4 369	4 460.3	0.15
	64	4 716	4 753.3	31.72	**4 662**	4 703.7	168.38	4 819	4 849.9	10.65	4 799	4 830.5	10.88	5 121	5 151.5	26.28	4 891	4 923.0	14.83	4 973	4 973	1.08	5 118	5 191.7	0.18
	128	5 761	5 805.8	22.80	**5 694**	5 731.4	73.69	5 843	5 875.3	16.83	5 838	5 862.8	16.97	6 204	6 291.5	36.66	5 801	5 846.3	17.85	5 951	5 951	1.24	5 990	6 074.4	0.21
ibm04	2	590	605.8	3.73	589	603.5	10.14	625	672.8	0.47	614	637.0	0.89	581	585.5	2.82	**580**	583.0	3.22	628	628	0.30	599	634.4	0.05
	4	1 760	1 782.0	9.81	1 740	1 762.8	32.74	1 831	1 880.7	1.05	1 796	1 834.6	1.47	1 651	1 673.2	4.08	**1 692**	1 711.4	6.53	1 815	1 815	0.55	1 830	1 906.0	0.09
	8	2 859	2 915.9	18.58	2 823	2 884.3	81.29	3 043	3 119.0	2.18	2 940	3 024.0	2.57	**2 808**	2 859.2	6.48	2 850	2 909.0	9.37	3 009	3 009	0.75	3 165	3 269.8	0.12
	16	3 777	3 867.6	27.40	**3 725**	3 807.6	138.82	3 975	4 098.0	4.07	3 884	4 003.3	4.45	3 882	3 946.3	10.33	4 031	4 058.1	12.31	4 035	4 035	0.92	4 217	4 346.5	0.15
	32	4 964	5 043.5	33.12	**4 857**	4 945.4	215.30	5 183	5 212.5	7.24	5 077	5 138.3	7.56	5 072	5 095.2	17.78	5 128	5 165.2	14.99	5 342	5 342	1.23	5 456	5 572.0	0.18
	64	6 086	6 133.0	36.05	**5 989**	6 030.2	229.76	6 214	6 284.9	12.30	6 169	6 226.2	12.57	6 445	6 507.2	28.23	6 329	6 387.2	17.98	6 496	6 496	1.40	6 652	6 744.0	0.22
	128	7 319	7 372.6	32.67	**7 225**	7 260.0	146.81	7 444	7 526.5	19.17	7 427	7 504.6	19.35	7 916	7 971.0	40.62	7 564	7 639.6	22.02	7 683	7 683	1.57	7 811	7 946.3	0.25
ibm05	2	1 726	1 734.9	16.11	1 726	1 731.2	40.93	1 770	1 789.5	0.74	1 742	1 752.8	1.33	1 726	1 729.7	6.11	**1 721**	1 726.3	6.81	1 730	1 730	0.28	1 740	1 757.8	0.07
	4	2 983	3 022.8	29.99	**2 945**	2 997.1	128.92	3 166	3 209.9	1.49	3 081	3 125.6	2.05	3 045	3 065.6	8.82	3 091	3 113.7	11.29	3 145	3 145	0.64	3 162	3 211.5	0.12
	8	4 320	4 562.3	48.26	**4 247**	4 490.7	263.99	4 724	4 872.6	2.79	4 559	4 767.2	3.32	4 408	4 486.5	12.84	4 489	4 553.5	14.03	4 698	4 698	0.86	4 764	4 884.8	0.15
	16	5 317	5 406.5	67.29	**5 140**	5 299.3	471.96	5 473	5 652.8	4.91	5 425	5 574.5	5.37	5 769	5 870.3	18.56	5 472	5 543.5	16.42	5 519	5 519	0.98	5 631	5 798.5	0.17
	32	5 907	5 998.7	79.07	**5 818**	5 902.0	666.01	6 029	6 133.9	8.16	5 962	6 083.1	8.54	6 703	6 814.7	27.63	6 255	6 347.3	19.11	6 637	6 637	1.15	6 265	6 443.0	0.20
	64	6 455	6 536.7	77.17	**6 335**	6 434.9	653.17	6 573	6 687.1	14.29	6 554	6 649.2	14.63	7 646	7 670.7	42.89	6 887	7 048.5	21.89	6 895	6 895	1.18	6 914	7 018.9	0.22
	128	7 126	7 224.2	51.32	**7 058**	7 140.1	291.01	7 277	7 395.2	21.50	7 251	7 373.7	21.95	8 691	8 747.6	57.79	7 479	7 580.8	24.36	7 300	7 300	1.32	7 418	7 525.3	0.25
ibm06	2	1 043	1 068.2	12.45	1 039	1 063.3	32.85	1 058	1 096.0	0.63	1 044	1 076.5	1.20	984	998.7	3.92	**981**	988.8	4.67	1 051	1 051	0.38	990	1 058.6	0.07
	4	1 507	1 637.9	20.22	**1 468**	1 623.5	69.94	1 564	1 688.3	1.24	1 549	1 656.9	1.78	1 495	1 515.5	5.58	1 696	1 727.8	8.46	1 546	1 546	0.59	1 573	1 826.4	0.11
	8	2 365	2 391.5	30.31	**2 333**	2 370.9	113.30	2 468	2 499.1	2.38	2 421	2 454.9	2.89	2 408	2 419.4	8.60	2 435	2 457.2	11.24	2 530	2 530	0.84	2 482	2 582.6	0.14
	16	3 210	3 253.9	41.71	**3 170**	3 204.0	237.87	3 381	3 413.3	4.43	3 305	3 349.3	4.90	3 262	3 336.9	13.54	3 370	3 379.8	14.11	3 421	3 421	1.00	3 472	3 624.9	0.18
	32	4 184	4 233.2	46.62	**4 137**	4 180.2	273.34	4 343	4 401.6	7.94	4 280	4 345.3	8.37	4 433	4 504.3	21.68	4 341	4 394.3	17.35	4 544	4 544	1.42	4 561	4 692.0	0.23
	64	5 124	5 171.3	50.59	**5 071**	5 117.0	268.70	5 324	5 354.0	13.49	5 276	5 309.8	13.90	5 720	5 759.4	37.38	5 317	5 359.6	21.27	5 487	5 487	1.38	5 648	5 706.0	0.26
	128	6 326	6 350.1	51.26	**6 213**	6 237.7	313.52	6 441	6 464.6	20.62	6 412	6 442.9	20.87	7 206	7 232.4	53.47	6 456	6 502.5	24.91	6 613	6 613	1.65	6 702	6 768.3	0.30
ibm07	2	935	967.1	9.02	**903**	955.0	22.62	1 025	1 059.7	0.75	997	1 032.0	1.52	953	970.5	5.63	925	950.8	6.66	1 002	1 002	0.37	997	1 028.4	0.08
	4	2 260	2 295.6	22.45	**2 173**	2 267.9	96.69	2 364	2 468.6	1.53	2 308	2 398.3	2.30	2 235	2 256.7	7.63	2 216	2 258.7	12.55	2 364	2 364	0.71	2 380	2 524.4	0.14
	8	3 386	3 451.1	35.46	**3 360**	3 418.5	154.26	3 530	3 648.4	2.87	3 483	3 575.1	3.62	3 366	3 434.7	10.71	3 459	3 513.9	16.22	3 584	3 584	1.17	3 780	3 908.8	0.20
	16	4 596	4 783.4	49.46	**4 561**	4 718.6	267.61	4 953	5 109.9	5.28	4 826	4 986.8	5.99	4 744	4 793.6	16.07	4 761	4 791.9	21.52	5 147	5 147	1.42	5 210	5 353.0	0.25
	32	5 980	6 073.0	59.91	**5 908**	5 997.2	377.63	6 242	6 347.7	9.41	6 142	6 256.4	10.07	6 359	6 409.8	25.78	6 378	6 416.2	25.56	6 488	6 488	1.70	6 754	6 871.7	0.31
	64	7 513	7 580.8	70.15	**7 416**	7 491.9	481.87	7 770	7 852.8	16.70	7 684	7 765.4	17.30	8 056	8 136.3	41.33	7 940	7 997.9	30.78	8 196	8 196	2.03	8 326	8 396.9	0.36
	128	9 123	9 190.2	77.41	**8 990**	9 050.9	527.13	9 385	9 426.7	25.26	9 346	9 380.9	25.56	10 126	10 201.9	60.66	9 570	9 636.4	35.39	9 826	9 826	2.42	9 914	10 024.1	0.41
ibm08	2	1 145	1 166.5	10.35	**1 141**	1 159.1	29.42	1 237	1 244.9	0.92	1 218	1 226.4	2.08	1 144	1 151.4	6.76	**1 141**	1 145.0	9.05	1 165	1 165	0.46	1 157	1 220.1	0.10
	4	2 343	2 369.0	24.27	**2 332**	2 352.3	91.50	2 500	2 531.6	1.63	2 453	2 466.7	2.77	2 382	2 407.4	9.92	2 468	2 475.6	15.85	2 470	2 470	0.94	2 510	2 627.6	0.18
	8	3 491	3 539.5	42.32	**3 455**	3 508.8	233.00	3 674	3 782.5	2.86	3 598	3 669.4	4.00	3 539	3 588.1	13.87	3 679	3 703.6	21.80	3 735	3 735	1.36	3 785	3 982.8	0.25
	16	4 572	4 643.0	64.79	**4 507**	4 564.1	442.46	4 939	5 023.0	5.06	4 819	4 893.5	6.18	4 687	4 832.9	19.69	4 888	4 954.3	26.81	5 090	5 090	1.72	5 111	5 260.1	0.31
	32	6 003	6 069.0	99.34	**5 878**	5 966.2	707.78	6 320	6 398.2	9.43	6 211	6 273.5	10.48	6 307	6 355.7	30.41	6 329	6 428.0	32.33	6 684	6 684	2.16	6 510	6 705.6	0.37
	64	7 817	7 888.3	147.22	**7 684**	7 753.9	1 258.31	8 026	8 100.0	17.01	7 935	8 016.4	17.96	8 301	8 333.3	48.88	8 259	8 330.4	36.68	8 320	8 320	2.39	8 480	8 566.7	0.43
	128	9 447	9 501.0	206.64	**9 366**	9 395.4	1 721.14	9 606	9 637.7	26.60	9 567	9 592.4	27.53	10 250	10 363.0	68.22	9 971	10 039.3	41.49	10 035	10 035	2.64	10 092	10 232.6	0.48
ibm09	2	620	624.2	6.19	**620**	621.7	15.05	642	649.3	0.73	629	635.9	1.71	627	632.1	5.08	627	631.6	5.39	638	638	0.49	639	683.9	0.10
	4	1 696	1 724.2	19.77	**1 690**	1 715.4	54.63	1 785	1 866.8	1.26	1 736	1 817.1	2.23	1 700	1 726.6	6.65	1 734	1 746.0	11.29	1 868	1 868	0.89	1 821	2 178.8	0.16
	8	2 682	2 837.6	31.03	2 682	2 821.0	82.12	2 851	2 977.4	2.33	2 773	2 921.6	3.28	**2 658**	2 723.8	8.96	2 719	2 737.7	15.79	2 953	2 953	1.22	3 024	3 199.8	0.23
	16	3 931	4 006.4	44.63	3 891	3 973.9	184.51	4 147	4 237.8	4.53	4 056	4 146.7	5.46	**3 886**	3 967.7	13.33	3 982	4 016.1	20.96	4 201	4 201	1.57	4 228	4 610.7	0.29
	32	5 476	5 542.2	60.92	**5 410**	5 477.8	275.78	5 625	5 753.1	8.79	5 542	5 646.8	9.63	5 447	5 496.7	22.85	5 503	5 564.0	26.98	5 803	5 803	2.06	6 124	6 189.1	0.36
	64	7 309	7 375.2	76.57	**7 209**	7 266.6	542.60	7 597	7 650.6	16.39	7 453	7 530.1	17.05	7 483	7 520.9	39.77	7 445	7 527.5	32.99	7 864	7 864	2.27	8 158	8 327.0	0.43
	128	9 383	9 483.1	85.30	**9 246**	9 350.9	573.46	9 665	9 725.5	26.12	9 600	9 659.3	26.52	9 955	10 007.3	59.20	9 843	9 923.8	39.40	10 006	10 006	2.79	10 348	10 513.1	0.49

H	k	KaHyPar-Strong			KaHyPar-StrongV			KaHyPar-Fast			KaHyPar-FastV			hMetis-K			hMetis-R			PaToH-Q			PaToH-D		
		min	avg	T[s]	min	avg	T[s]	min	avg	T[S]	min	avg	T[s]	min	avg	T[s]	min	avg	T[s]	min	avg	T[s]	min	avg	T[s]
ibm10	2	1 293	1 382.8	14.96	**1 288**	1 381.4	31.30	1 411	1 520.5	1.09	1 332	1 439.6	2.47	1 317	1 340.0	9.54	1 328	1 340.7	10.44	1 658	1 658	0.73	1 517	1 763.6	0.14
	4	2 414	2 513.3	25.23	**2 345**	2 464.6	91.25	2 706	2 870.4	1.84	2 590	2 741.0	3.21	2 423	2 488.8	11.84	2 481	2 551.3	20.45	2 861	2 861	1.25	2 775	3 139.7	0.25
	8	4 225	4 299.0	44.32	4 128	4 218.4	209.78	4 496	4 685.3	3.40	4 363	4 508.5	4.75	**4 056**	4 136.8	16.65	4 216	4 298.0	28.35	4 658	4 658	1.84	4 463	4 939.3	0.34
	16	6 019	6 258.3	71.34	5 964	6 182.3	382.71	6 330	6 620.3	6.38	6 244	6 458.7	7.68	**5 884**	6 024.1	22.86	6 151	6 275.1	35.83	6 667	6 667	2.31	6 597	6 867.6	0.43
	32	8 552	8 626.3	97.26	**8 377**	8 508.6	575.48	9 114	9 235.7	11.88	8 919	9 024.8	13.12	8 531	8 634.5	35.36	8 638	8 799.5	44.20	8 988	8 988	2.98	9 197	9 479.4	0.53
	64	11 722	11 791.7	134.69	**11 407**	11 530.4	1 048.91	12 198	12 289.3	21.31	11 991	12 072.7	22.45	11 837	11 918.8	56.46	11 781	11 883.5	51.55	12 303	12 303	3.31	12 559	12 709.9	0.62
	128	14 662	14 779.8	161.68	**14 356**	14 467.4	1 232.47	15 054	15 222.5	33.36	14 847	15 037.6	34.46	15 234	15 294.6	85.60	15 000	15 061.3	59.78	15 254	15 254	3.97	15 744	15 867.1	0.71
ibm11	2	1 077	1 170.5	11.94	1 071	1 152.4	37.72	1 108	1 240.6	0.97	1 085	1 205.5	2.29	**1 063**	1 068.8	7.11	1 065	1 067.7	8.02	1 105	1 105	0.57	1 078	1 184.9	0.12
	4	2 455	2 533.0	25.58	**2 437**	2 517.2	65.42	2 614	2 694.4	1.64	2 541	2 618.8	2.95	2 484	2 513.8	9.33	2 492	2 530.5	15.50	2 755	2 755	1.13	2 810	3 038.2	0.21
	8	3 552	3 796.6	41.35	**3 502**	3 736.0	165.37	3 774	3 962.4	2.99	3 611	3 866.5	4.29	3 572	3 644.7	11.96	3 750	3 793.4	22.51	3 826	3 826	1.48	4 024	4 253.3	0.29
	16	5 134	5 350.7	60.43	**5 057**	5 284.7	253.74	5 385	5 642.2	5.70	5 244	5 481.5	6.96	5 276	5 349.9	17.40	5 301	5 540.0	29.17	5 608	5 608	2.02	5 813	6 071.7	0.37
	32	7 357	7 502.4	82.01	**7 223**	7 398.0	490.42	7 660	7 865.8	10.55	7 510	7 684.5	11.76	7 536	7 595.2	28.34	7 600	7 634.9	35.87	7 841	7 841	2.59	8 179	8 363.6	0.46
	64	9 632	9 722.8	103.31	**9 512**	9 578.7	673.44	10 066	10 114.5	19.16	9 867	9 946.6	20.36	9 907	10 004.1	46.51	9 809	10 013.9	43.55	10 284	10 284	3.16	10 778	10 938.5	0.54
	128	12 775	12 819.5	122.85	**12 610**	12 647.6	872.82	13 192	13 261.2	30.22	13 056	13 115.3	31.31	13 372	13 441.3	71.95	13 131	13 230.4	50.79	13 478	13 478	3.65	13 884	14 197.8	0.63
ibm12	2	2 040	2 056.2	20.72	2 031	2 045.9	59.92	2 135	2 191.4	1.18	2 101	2 130.3	2.69	1 995	2 052.3	13.14	**1 951**	1 969.0	15.14	1 988	1 988	0.74	2 071	2 180.8	0.16
	4	3 829	4 119.0	42.47	**3 796**	4 087.2	166.43	4 163	4 404.3	2.01	3 925	4 282.1	3.53	3 994	4 046.7	14.84	3 918	3 984.7	24.89	4 848	4 848	1.35	4 323	4 603.4	0.27
	8	6 211	6 312.3	65.98	6 137	6 242.2	367.19	6 524	6 664.3	3.72	6 406	6 508.5	5.23	**5 999**	6 135.3	17.70	6 139	6 235.6	33.33	6 287	6 287	1.97	6 641	7 048.6	0.38
	16	8 559	8 781.0	91.85	8 427	8 678.0	540.58	8 973	9 310.0	6.34	8 759	9 086.3	7.77	**8 256**	8 336.0	23.97	8 374	8 489.7	43.83	9 098	9 098	2.37	9 426	9 687.8	0.48
	32	10 683	10 942.0	111.49	**10 530**	10 805.0	716.21	11 245	11 517	11.96	11 006	11 286.9	13.40	10 820	10 939.3	37.38	11 049	11 155.1	51.46	11 676	11 676	3.14	11 956	12 331.6	0.58
	64	14 178	14 331.3	141.69	**13 946**	14 126.5	1 049.58	14 793	14 961.1	22.77	14 568	14 727.5	24.17	14 525	14 635.8	60.06	14 551	14 755.3	59.03	15 335	15 335	3.74	15 979	16 210.0	0.67
	128	18 186	18 262.5	180.00	**17 845**	17 962.2	1 390.90	18 780	18 859.9	36.37	18 605	18 695.3	37.78	19 270	19 358.4	93.87	18 676	18 832.8	70.08	19 162	19 162	4.34	19 639	19 888.5	0.76
ibm13	2	832	832.2	14.73	**832**	832.1	25.92	852	868.7	1.24	845	857.6	2.95	833	843.4	11.26	837	850.4	10.52	891	891	0.70	877	1 134.5	0.16
	4	1 943	2 082.3	32.81	1 931	2 048.6	119.14	2 057	2 429.4	1.99	2 007	2 317.1	3.79	**1 845**	1 901.3	13.15	1 953	1 971.3	20.01	2 182	2 182	1.27	2 016	2 402.2	0.28
	8	2 977	3 164.9	49.54	2 966	3 114.6	180.25	3 248	3 518.7	3.54	3 154	3 391.9	5.30	**2 891**	3 012.2	17.10	3 053	3 094.5	28.35	3 718	3 718	2.05	3 245	3 867.6	0.39
	16	5 610	5 753.9	84.04	**5 404**	5 644.2	396.09	5 965	6 048.8	6.82	5 784	5 866.0	8.55	5 541	5 599.4	24.46	5 610	5 658.2	38.34	5 968	5 968	2.85	6 156	6 560.2	0.50
	32	7 849	7 943.0	114.12	**7 686**	7 799.5	664.11	8 246	8 350.6	12.61	8 060	8 138.2	14.33	7 696	7 848.0	38.06	7 743	7 903.8	48.26	8 578	8 578	3.26	8 406	8 820.5	0.61
	64	12 019	12 057.8	158.19	**11 830**	11 872.2	1 204.50	12 386	12 465.2	23.18	12 205	12 277.0	24.73	12 159	12 243.5	61.83	12 161	12 254.3	57.84	12 892	12 892	3.94	13 201	13 380.5	0.72
	128	15 446	15 503	197.26	**15 246**	15 326.6	1 592.22	15 885	15 968.2	35.30	15 741	15 808.5	36.71	16 248	16 326.3	90.36	15 707	15 822.0	66.48	16 486	16 486	4.78	16 835	17 051.3	0.82
ibm14	2	1 896	1 926.4	39.28	1 888	1 917.7	117.61	2 001	2 057.9	2.21	1 979	2 033.3	5.61	1 874	1 911.1	21.05	**1 869**	1 881.2	23.80	2 107	2 107	1.26	2 086	2 292.7	0.28
	4	3 345	3 426.4	70.15	**3 307**	3 357.3	294.00	3 594	3 792.0	3.19	3 468	3 635.5	6.59	3 370	3 436.8	26.24	3 379	3 433.2	40.62	3 568	3 568	2.23	3 541	3 981.2	0.50
	8	5 188	5 301.5	115.84	5 139	5 224.1	598.93	5 363	5 809.2	5.34	5 289	5 525.5	8.71	5 078	5 185.3	31.15	**5 070**	5 295.5	59.06	5 879	5 879	3.28	5 688	6 151.5	0.70
	16	8 290	8 534.6	176.74	**8 114**	8 380.2	1 140.84	8 923	9 086.9	9.71	8 630	8 831.5	13.04	8 360	8 460.7	40.37	8 451	8 531.8	74.65	9 050	9 050	4.53	9 186	9 529.7	0.88
	32	12 837	13 073.4	268.11	**12 576**	12 830.0	1 940.90	13 683	13 940.8	17.48	13 342	13 552.4	20.80	12 735	12 878.2	56.96	12 759	12 890.7	89.16	13 666	13 666	5.55	13 805	14 253.2	1.07
	64	17 478	17 668.0	350.12	**17 212**	17 356.5	2 783.16	18 412	18 554.2	30.78	18 039	18 167.7	34.00	17 823	17 917.1	87.22	17 781	17 857.7	104.04	18 573	18 573	6.64	19 111	19 332.5	1.23
	128	22 222	22 439.3	410.60	**21 837**	22 043.0	3 421.50	23 180	23 387.7	47.28	22 820	22 982.1	50.28	23 243	23 412.9	125.75	22 990	23 141.0	117.75	23 647	23 647	7.61	24 331	24 605.2	1.39
ibm15	2	2 746	2 762.7	46.19	**2 744**	2 752.9	109.66	2 785	2 830.5	2.93	2 766	2 778.7	7.63	2 781	2 837.3	28.91	**2 744**	2 808.6	29.75	2 768	2 768	1.56	2 790	3 149.2	0.35
	4	5 042	5 166.5	88.79	5 019	5 134.2	293.56	5 274	5 426.0	3.97	5 191	5 333.4	8.72	4 836	4 911.2	33.44	**4 825**	4 984.8	52.91	5 245	5 245	2.61	5 599	5 970.5	0.63
	8	6 808	6 979.2	132.10	6 614	6 875.9	542.95	7 046	7 303.6	6.12	6 952	7 152.7	10.83	**6 523**	6 741.4	38.98	6 639	6 740.3	71.35	7 735	7 735	3.97	7 772	8 430.8	0.87
	16	8 769	9 204.3	189.96	**8 726**	9 093.2	910.88	9 242	9 727.6	10.34	8 962	9 507.8	15.03	8 743	8 923.7	49.68	8 740	9 043.2	90.86	9 930	9 930	5.08	10 117	10 765.2	1.07
	32	13 801	13 980.9	292.86	13 642	13 771.7	1 788.66	14 382	14 614.2	18.70	14 100	14 292.1	23.25	13 477	13 729.1	69.06	**13 441**	13 604.1	110.93	14 927	14 927	6.38	15 069	15 671.5	1.28
	64	18 608	18 907.2	397.94	**18 303**	18 631.8	3 009.40	19 518	19 814.0	33.79	19 102	19 406.2	38.26	18 949	19 101.7	102.89	18 768	18 994.7	128.36	19 793	19 793	7.75	20 327	21 040.0	1.51
	128	24 954	25 204.9	504.85	**24 586**	24 829.7	3 999.80	26 338	26 477.6	53.46	25 815	26 001.3	57.36	25 574	25 715.0	147.45	25 192	25 305.9	149.80	26 720	26 720	8.96	27 554	27 858.8	1.72
ibm16	2	1 981	2 083.5	41.52	1 981	2 067.4	97.93	2 102	2 171.8	3.14	2 030	2 134.9	8.31	2 006	2 046.8	30.84	**1 913**	1 978.5	34.61	2 080	2 080	1.30	2 128	2 314.3	0.42
	4	4 255	4 379.0	91.05	4 211	4 343.0	307.62	4 403	4 563.9	4.31	4 274	4 458.4	9.51	4 175	4 276.6	38.70	**4 180**	4 261.8	63.24	4 622	4 622	3.37	4 586	4 986.1	0.74
	8	6 710	6 986.9	147.54	**6 662**	6 907.4	740.28	7 213	7 439.0	6.54	7 094	7 265.6	11.71	6 737	6 903.8	43.75	6 927	6 982.3	87.30	7 911	7 911	4.84	7 860	8 237.4	1.03
	16	10 771	11 014.6	248.16	**10 634**	10 917.5	1 364.47	11 546	11 781.5	11.34	11 253	11 447.2	16.54	10 783	10 975.9	54.81	10 859	11 007.3	108.34	11 379	11 379	5.94	11 724	12 324.1	1.28
	32	15 202	15 449.4	368.11	**15 066**	15 245.5	2 384.34	16 124	16 409.2	19.69	15 721	15 984.1	24.89	15 493	15 638.1	73.34	15 562	15 777.9	128.19	16 464	16 464	7.37	17 030	17 371.5	1.52
	64	20 477	20 660.3	511.35	**20 162**	20 368.6	4 097.76	21 470	21 642.2	34.40	20 987	21 204.6	39.43	20 969	21 124.2	106.24	20 939	21 193.5	147.66	21 937	21 937	8.59	22 734	23 133.0	1.75
	128	26 279	26 464.2	631.24	**25 928**	26 117.7	5 239.48	27 196	27 469.3	53.91	26 780	27 059.3	58.57	27 602	27 784.0	152.48	27 223	27 444.0	167.45	28 115	28 115	9.72	28 983	29 347.1	1.97
ibm17	2	2 373	2 498.0	65.08	2 359	2 483.0	175.76	2 506	2 886.9	3.62	2 467	2 707.5	9.35	2 318	2 354.4	42.23	**2 317**	2 343.7	46.42	2 535	2 535	2.03	2 462	2 721.8	0.47
	4	5 611	5 933.3	136.60	**5 587**	5 874.1	585.92	6 200	6 737.2	5.01	6 012	6 387.8	10.79	5 685	5 900.3	52.25	5 665	5 727.9	82.20	5 783	5 783	3.57	6 388	7 024.5	0.85
	8	9 589	9 919.6	220.52	**9 516**	9 786.4	1 381.10	10 272	10 710.9	7.78	9 932	10 302.5	13.56	9 551	9 897.6	59.47	9 654	9 775.6	116.90	10 142	10 142	5.35	10 691	11 805.1	1.18
	16	14 501	14 926.1	343.07	**14 281**	14 676.4	2 247.49	15 605	16 082.8	13.36	15 103	15 557.2	19.08	14 734	15 086.7	71.92	15 080	15 313.0	146.84	16 543	16 543	7.01	16 125	17 402.6	1.47
	32	19 651	19 962.4	456.00	**19 272**	19 617.9	3 270.54	20 742	21 253.3	22.90	20 289	20 648.0	28.62	19 776	19 936.2	94.44	20 105	20 303.1	166.76	20 886	20 886	8.62	21 811	23 072.8	1.75
	64	26 496	26 910.2	662.70	26 149	26 462.0	5 152.58	28 180	28 482.0	40.43	27 571	27 785.0	45.96	**26 051**	26 237.3	134.78	26 505	26 792.6	185.28	29 612	29 612	10.14	29 665	30 102.9	1.99
	128	34 524	34 808.3	808.83	34 009	34 202.0	6 657.54	36 103	36 499.7	64.79	35 455	35 839.5	69.89	35 531	35 617.8	192.30	35 274	35 571.8	210.12	37 172	37 172	11.39	38 086	38 546.3	2.22
ibm18	2	1 542	1 750.8	32.98	**1 529**	1 722.8	117.75	1 694	1 911.7	3.80	1 664	1 871.2	10.20	1 873	1 983.2	37.12	1 683	1 878.7	40.41	2 001	2 001	1.78	1 739	2 054.4	0.46
	4	3 098	3 215.8	61.79	**3 027**	3 143.8	247.84	3 324	3 448.9	4.98	3 240	3 374.0	11.47	3 111	3 208.8	46.38	3 191	3 228.5	77.44	3 950	3 950	3.29	3 866	4 150.0	0.81
	8	5 826	6 034.4	131.84	**5 742**	5 911.3	829.24	6 359	6 589.2	7.58	6 102	6 359.6	14.16	5 773	6 098.6	52.15	5 934	6 111.5	102.37	6 582	6 582	4.68	6 817	7 168.1	1.11
	16	8 701	8 855.6	214.33	**8 539**	8 690.6	1 472.97	9 374	9 556.0	12.39	9 055	9 247.6	18.87	8 907	9 094.8	65.77	9 027	9 175.2	126.69	9 576	9 576	5.94	9 796	10 285.8	1.38
	32	13 013	13 324.0	362.17	**12 821**	13 059.4	2 977.57	13 970	14 228.7	21.04	13 669	13 849.4	27.51	13 453	13 651.3	88.16	13 479	13 853.2	151.33	14 060	14 060	7.48	14 567	14 946.9	1.64
	64	18 315	18 505.5	576.36	**17 946**	18 186.0	5 114.56	19 321	19 535.6	36.17	18 902	19 105.3	42.31	18 656	18 822.1	126.46	18 790	19 058.1	171.99	19 571	19 571	8.91	20 018	20 491.1	1.87
	128	24 156	24 258.6	884.23	**23 682**	23 821.7	7 742.10	25 144	25 342.2	57.97	24 791	24 874.7	64.24	25 285	25 391.5	176.25	24 773	24 911.4	191.64	25 584	25 584	10.17	25 887	26 294.4	2.10
G3_circuit	2	**2 142**	2 209.2	113.99	**2 142**	2 209.2	225.30	2 346	2 537.0	93.40	2 304	2 498.9	280.25	2 326	2 412.8	282.62	**2 142**	**2 142.0**	258.41	2 430	2 430	9.74	2 208	2 355.2	2.15
	4	5 352	5 484.4	144.86	5 350	5 477.2	451.93	5 984	6 157.7	93.73	5 913	6 079.6	281.04	5 778	5 927.6	312.77	**5 278**	**5 278.0**	488.09	6 640	6 640	15.21	5 680	6 268.8	3.87
	8	9 689	9 847.5	188.27	9 680	9 835.2	635.47	10 707	11 007.1	94.39	10 583	10 860.7	282.14	10 394	10 635.7	328.02	**9 609**	9 685.4	657.37	10 471	10 471	20.22	10 546	11 002.0	5.35
	16	15 314	15 953.4	256.11	**15 284**	15 901.6	1 109.05	17 525	18 047.3	95.87	17 349	17 824.3	283.92	17 314	17 453.6	336.14	15 951	16 027.4	819.39	17 847	17 847	26.85	17 358	18 374.9	6.68
	32	23 193	23 781.7	349.99	**23 152**	23 722.6	1 934.72	26 115	26 563.9	99.43	25 672	26 187.4	288.26	25 584	26 043.4	347.24	23 917	24 199.0	957.99	25 967	25 967	37.99	26 286	27 726.2	7.96
	64	36 802	37 561.0	512.26	**36 713**	37 454.7	3 395.71	40 814	41 672.9	109.02	40 225	41 041.4	297.98	40 243	40 573.0	372.30	37 935	38 199.1	1 050.94	41 219	41 219	38.08	40 717	42 387.1	9.24
	128	61 391	61 873.3	803.93	**60 974**	61 503.6	6 963.77	66 886	67 216.5	129.14	65 872	66 181.8	318.89	65 227	65 704.9	421.42	63 095	63 867.4	1 193.03	66 471	66 471	52.01	67 727	68 680.7	10.64

		KaHyPar-Strong			KaHyPar-StrongV			KaHyPar-Fast			KaHyPar-FastV			hMetis-K			hMetis-R			PaToH-Q			PaToH-D		
H	k	min	avg	T[s]	min	avg	T[s]	min	avg	T[S]	min	avg	T[s]	min	avg	T[s]	min	avg	T[s]	min	avg	T[s]	min	avg	T[s]
af_shell10	2	**5 250**	**5 250.0**	120.82	**5 250**	**5 250.0**	245.43	5 430	5 632.0	101.39	5 310	5 428.0	311.69	5 250	5 294.0	382.75	**5 250**	**5 250.0**	359.60	5 290	5 290	21.47	5 250	5 392.0	10.57
	4	10 930	11 218.5	146.55	10 930	11 217.5	309.78	12 310	12 659.5	101.66	11 920	12 186.5	312.27	11 335	11 614.0	466.23	**10 905**	11 043.0	688.74	11 840	11 840	40.62	11 495	11 879.5	20.57
	8	20 805	21 156.0	189.23	20 785	21 140.5	551.28	23 285	23 521.5	102.24	22 525	22 694.5	315.37	21 440	21 694.0	527.40	**20 655**	21 340.5	973.74	21 595	21 595	64.59	21 545	22 617.5	30.28
	16	31 655	32 450.0	243.56	**31 645**	32 425.5	798.23	36 415	36 848.0	103.43	35 125	35 476.5	315.01	33 990	34 218.5	551.03	33 050	33 530.0	1 258.71	33 690	33 690	78.21	35 105	35 834.0	39.81
	32	50 645	51 550.0	335.61	**50 550**	51 508.5	1 233.99	57 075	57 615.0	106.11	54 995	55 601.5	318.11	52 340	53 208.0	560.08	51 995	52 919.0	1 512.51	53 340	53 340	98.12	54 020	55 063.5	49.17
	64	73 315	75 115.0	462.17	**73 185**	75 020.5	1 802.97	83 845	84 219.5	111.51	80 920	81 408.0	324.57	78 030	78 754.5	571.85	76 905	77 646.0	1 751.95	78 110	78 110	118.82	81 065	81 562.0	58.40
	128	110 815	111 676.5	668.61	**110 575**	111 527.5	3 300.43	122 965	123 513	122.84	119 305	119 689.0	336.31	112 770	114 015	606.54	114 345	114 926.5	1 982.89	115 410	115 410	138.47	118 340	118 801.5	67.56
af_shell9	2	1 810	1 876.0	19.60	1 810	1 876.0	41.10	2 020	2 087.0	16.54	1 955	2 017.0	50.59	1 850	1 896.0	111.56	1 770	**1 770.0**	102.96	1 855	1 855	6.13	1 830	1 911.0	2.46
	4	**4 370**	4 449.0	24.78	**4 370**	4 449.0	50.95	4 780	4 907.5	16.54	4 650	4 740.0	52.68	4 495	4 535.0	116.57	**4 370**	4 380.0	194.86	4 550	4 550	11.87	4 500	4 719.0	4.72
	8	**8 325**	8 559.5	34.15	**8 325**	8 552.5	75.38	9 530	9 706.5	16.98	9 185	9 347.5	53.21	8 830	8 903.0	145.23	8 515	8 540.5	286.05	8 825	8 825	17.52	8 915	9 321.0	6.94
	16	15 675	16 032.0	51.89	**15 675**	16 013.0	153.78	17 785	18 084.5	17.88	17 170	17 435.5	54.15	16 410	16 616.0	154.85	16 190	16 260.5	371.48	16 700	16 700	27.20	16 915	17 353.5	9.12
	32	25 900	26 344.5	80.61	**25 825**	26 328.5	210.36	29 085	29 413.5	19.86	28 275	28 473.0	56.33	26 850	27 113.5	164.40	26 800	26 957.5	452.62	27 250	27 250	32.65	27 800	28 225.5	11.28
	64	40 585	40 816.0	127.71	**40 425**	40 736.0	603.20	44 885	45 202.5	25.43	43 455	43 858.0	61.91	41 850	42 005.5	176.20	41 560	41 805.0	531.32	42 580	42 580	36.31	43 355	43 647.5	13.42
	128	60 495	60 725.0	205.52	**60 290**	60 518.5	1 348.21	66 315	66 555.0	37.46	64 795	65 039.0	73.59	61 405	61 782.5	206.49	62 220	62 553.5	607.81	62 765	62 765	42.48	63 985	64 528.0	15.56
audikw_1	2	10 560	10 644.0	278.54	**10 509**	10 624.2	763.44	10 818	11 110.5	146.91	10 746	11 028.0	465.01	10 914	11 083.2	676.12	10 986	11 140.2	865.87	10 860	10 860	97.02	10 962	11 199.3	37.39
	4	32 091	32 176.8	717.20	**32 025**	32 112.6	2 931.02	33 267	33 888.3	178.30	33 120	33 652.5	560.42	33 396	33 673.2	729.31	33 864	34 680.9	1 699.25	34 800	34 800	191.89	34 200	34 970.1	73.00
	8	74 868	75 132.0	1 589.63	**74 544**	74 802.0	12 402.15	77 403	78 483.6	238.93	76 962	77 792.7	739.58	77 688	78 567.6	741.06	80 445	81 373.5	2 433.03	79 962	79 962	280.54	80 547	82 917.0	106.80
	16	122 154	124 270.2	2 881.68	**121 689**	123 594.6	21 594.15	129 174	130 495.5	346.81	128 208	129 464.4	1 055.76	129 114	130 464.6	768.66	133 719	136 167.6	3 020.54	133 212	133 212	364.56	135 351	137 769.9	138.22
	32	181 608	183 928.5	4 476.22	**180 930**	182 999.4	30 219.78	190 335	192 516.9	502.39	188 943	190 809.6	1 489.48	190 656	192 657.6	819.98	200 793	202 715.1	3 517.65	196 476	196 476	442.18	200 628	203 876.4	167.05
	64	257 043	258 542.4	6 667.86	**255 102**	257 155.5	40 652.73	267 486	269 372.4	787.31	265 656	267 321.3	2 292.19	270 621	273 032.7	940.62	282 318	284 542.8	3 974.58	272 472	272 472	482.41	283 095	287 144.1	192.87
	128	357 429	358 326.9	9 958.66	**354 792**	356 290.8	53 424.83	368 904	370 119.9	1 308.07	367 131	368 009.7	3 731.34	374 292	377 195.1	1 164.06	385 416	387 915.0	4 324.49	375 576	375 576	568.18	388 149	389 672.4	215.12
ecology1	2	**2 000**	**2 000.0**	46.44	**2 000**	**2 000.0**	93.20	2 006	2 017.6	39.18	**2 000**	2 005.4	118.10	**2 000**	2 000.2	104.63	**2 000**	**2 000.0**	127.21	**2 000**	**2 000**	3.24	**2 000**	2 007.2	1.03
	4	3 694	3 748.2	54.39	3 694	3 746.8	124.69	3 933	3 972.3	39.48	3 883	3 931.9	118.46	3 842	3 877.4	144.87	3 667	3 777.7	234.12	3 938	3 938	7.60	3 745	3 977.5	1.81
	8	6 502	6 652.1	68.58	**6 439**	6 568.5	656.78	7 385	7 553.0	39.63	7 319	7 479.5	119.06	6 994	7 069.2	165.81	6 668	7 042.8	324.69	7 372	7 372	10.67	7 346	7 654.5	2.49
	16	10 193	10 300.8	87.67	**10 118**	10 253.4	452.49	11 579	11 692.6	40.67	11 438	11 558.8	120.03	11 283	11 365.9	173.90	10 559	10 955.2	397.33	11 182	11 182	13.91	11 748	11 914.1	3.14
	32	15 658	15 988.1	118.75	**15 474**	15 841.8	1 047.81	18 178	18 386.9	42.97	17 903	18 128.3	122.61	17 365	17 500.8	185.62	15 605	16 779.4	460.38	17 960	17 960	17.36	17 958	18 433.1	3.79
	64	22 637	22 914.5	161.38	**22 448**	22 664.6	1 457.42	26 296	26 540.1	48.50	25 798	26 105.4	127.59	25 315	25 594.8	197.76	23 216	23 875.5	529.98	25 247	25 247	20.94	26 517	27 073.4	4.47
	128	32 830	33 359.4	234.13	**32 444**	32 891.2	2 027.55	38 690	38 968.8	59.05	37 979	38 231.8	138.78	36 548	36 921.3	224.26	33 163	34 163.7	594.17	37 054	37 054	23.18	38 747	39 317.4	5.19
ecology2	2	**1 998**	1 999.6	46.66	**1 998**	1 999.4	97.86	2 004	2 023.6	39.28	2 000	2 009.4	114.23	2 000	2 000.0	106.11	**1 998**	1 998.6	129.93	2 000	2 000	3.34	2 000	2 024.4	1.04
	4	3 686	3 737.9	54.36	**3 683**	3 733.1	135.81	3 923	3 955.4	39.50	3 880	3 916.5	118.90	3 843	3 872.6	136.97	3 757	3 848.0	233.99	3 945	3 945	7.35	3 919	3 982.8	1.82
	8	6 531	6 638.1	68.33	**6 458**	6 573.3	634.30	7 449	7 587.5	39.81	7 367	7 512.9	119.11	7 040	7 102.3	153.97	6 717	7 249.3	324.76	7 671	7 671	8.78	7 090	7 610.0	2.50
	16	10 148	10 307.8	86.77	**10 088**	10 255.9	574.26	11 564	11 630.9	40.84	11 435	11 491.7	120.22	11 293	11 372.3	174.72	10 334	10 964.9	402.93	11 741	11 741	12.26	11 748	11 942.2	3.15
	32	15 449	15 998.2	118.12	**15 376**	15 861.7	1 089.90	18 057	18 374.4	43.03	17 752	18 119.9	122.46	17 099	17 404.1	183.80	15 831	16 546.1	473.67	18 366	18 366	17.10	18 017	18 382.6	3.81
	64	22 607	22 881.3	161.14	**22 282**	22 592.6	1 449.54	26 287	26 537.2	48.37	25 867	26 133.6	128.08	25 277	25 609.2	199.53	23 349	23 902.6	537.28	25 951	25 951	20.23	26 460	26 950.5	4.48
	128	33 034	33 317.4	233.79	**32 536**	32 836.4	2 015.19	38 386	38 829.1	59.06	37 570	38 069.7	139.27	36 351	36 938.5	225.02	33 510	34 196.3	593.44	37 974	37 974	21.99	38 037	38 970.9	5.21
kkt_power	2	10 097	11 049.8	697.86	10 097	11 001.0	1 405.95	10 199	14 219.8	190.36	10 108	13 553.1	566.77	**8 104**	8 699.0	382.56	**8 104**	8 219.4	437.80	10 050	10 050	54.02	9 853	10 509.5	9.74
	4	17 291	18 983.0	1 113.03	**17 291**	18 697.7	3 325.62	19 357	23 953.9	195.24	19 041	23 344.6	573.94	18 616	19 437.0	413.41	17 538	17 981.7	766.20	23 246	23 246	62.85	18 923	21 236.7	16.21
	8	33 197	34 712.5	1 925.05	**32 725**	33 918.2	6 968.26	37 084	45 868.9	203.72	36 622	44 751.2	586.42	33 947	35 911.9	436.46	33 212	33 944.1	1 064.56	38 654	38 654	99.68	35 970	39 119.2	21.40
	16	57 239	59 985.4	3 212.94	56 725	58 333.8	12 684.93	64 462	73 127.6	214.36	64 134	72 023.3	598.95	59 227	63 293.4	550.10	**54 068**	54 696.8	1 311.94	64 547	64 547	145.94	61 163	68 433.8	25.66
	32	91 178	96 174.1	5 497.53	89 426	92 914.1	34 883.90	106 325	110 547.5	232.35	105 033	109 192.3	620.40	106 753	111 711.5	619.80	**86 610**	87 692.9	1 545.61	103 367	103 367	159.05	107 817	111 476.4	29.37
	64	142 021	147 434.6	8 919.73	137 438	142 421.8	69 123.78	152 865	161 682.4	264.50	150 982	159 922.1	655.42	163 635	172 896.8	708.97	**133 652**	136 666.4	1 727.55	153 857	153 857	182.53	164 257	170 326.1	32.32
	128	206 584	211 133.1	13 510.53	202 552	207 234.2	72 654.16	220 451	224 159.0	306.48	217 538	221 283.2	699.83	235 792	241 952.6	847.12	**201 856**	204 049.2	1 884.65	223 054	223 054	211.62	240 786	245 974.2	34.89
ldoor	2	3 136	3 204.6	64.45	**3 129**	3 200.4	165.32	3 437	3 549.7	54.93	3 367	3 459.4	173.26	3 276	3 327.8	327.59	3 206	3 273.9	353.13	3 584	3 584	23.44	3 381	3 520.3	12.75
	4	6 321	6 439.3	75.03	**6 300**	6 424.6	231.94	6 846	7 057.4	55.02	6 699	6 873.3	172.07	6 510	6 645.8	370.63	6 538	6 640.2	629.39	7 266	7 266	59.89	6 790	7 282.1	24.89
	8	11 424	11 868.5	94.25	11 312	11 823.7	381.70	12 460	13 075.3	55.40	12 229	12 791.8	173.86	**11 039**	11 777.5	384.03	11 921	12 149.2	903.98	12 222	12 222	79.74	12 607	13 221.6	36.66
	16	20 097	20 547.1	127.55	**20 083**	20 472.9	637.14	21 959	22 443.4	56.45	21 497	21 974.4	174.37	20 944	21 203.0	386.94	20 587	21 132.3	1 095.73	22 001	22 001	88.92	22 344	23 104.9	48.16
	32	34 188	34 901.3	188.20	**33 978**	34 587.0	999.94	37 177	38 098.2	58.96	36 442	37 384.2	176.96	34 797	35 707.7	393.46	35 595	35 980.7	1 276.01	36 862	36 862	111.35	37 198	38 202.5	59.42
	64	52 556	53 447.1	276.89	**52 381**	53 238.5	1 439.70	56 763	57 830.5	65.06	56 049	56 931.0	183.36	55 209	55 837.6	411.61	55 461	56 001.4	1 484.94	57 050	57 050	129.05	57 855	58 896.6	70.43
	128	80 052	81 370.8	422.62	**79 506**	80 864.0	3 030.51	86 212	86 853.9	77.43	85 113	85 726.9	195.74	84 168	84 784.7	441.99	83 860	84 725.2	1 629.32	86 359	86 359	158.38	87 087	88 189.5	81.19
nlpkkt120	2	**58 560**	60 125.3	2 875.14	**58 560**	59 551.9	10 336.76	72 049	74 427.6	623.71	69 521	72 256.5	1 885.06	70 606	71 812.6	5 618.75	**58 560**	**58 560.0**	4 735.45	**58 560**	**58 560**	94.77	60 208	64 489.8	32.81
	4	119 643	122 683.9	5 628.59	118 326	121 933.0	26 623.38	145 336	148 157.4	631.97	141 290	144 304.2	1 908.13	141 014	143 655.9	5 680.63	**116 152**	116 211.2	8 574.06	119 387	119 387	236.32	125 448	134 492.7	59.39
	8	179 944	183 170.3	8 638.67	181 754	183 444.8	42 291.50	216 526	220 191.5	642.59	210 008	214 201.9	1 931.20	210 257	214 035.5	5 720.41	**172 792**	172 944.2	11 617.46	174 576	174 576	348.70	184 414	195 314.1	83.48
	16	289 842	293 837.5	14 116.15	287 674	291 675.0	83 200.74	341 050	346 855.7	662.55	332 180	337 401.5	1 972.85	332 833	336 909.4	5 918.25	**286 134**	286 609.6	14 444.24	288 506	288 506	420.81	303 593	311 787.2	106.30
	32	408 345	413 522.1	20 729.70	407 879	412 334.0	130 031.00	470 624	477 911.4	700.46	457 573	464 404.7	2 026.20	458 401	465 345.2	5 918.25	**396 606**	397 339.2	16 197.80	403 292	403 292	524.99	421 811	433 883.7	127.39
	64	531 214	535 472.5	28 362.56	530 163	533 236.2	140 679.34	611 293	615 332.3	780.47	593 156	597 340.3	2 112.69	594 092	599 176.4	6 210.84	**503 760**	504 327.9	18 099.16	519 271	519 271	579.15	543 153	562 896.9	147.18
	128	724 650	731 340.9	40 514.36	719 453	725 664.5	144 816.50	813 483	821 307.1	975.02	790 269	797 385.4	2 320.81	795 836	801 331.9	6 702.57	**713 933**	714 740.7	19 430.12	721 912	721 912	649.47	748 049	757 780.6	166.38
cage15	2	-	-	-	-	-	-	627 557	631 461.2	1 725.2	626 499	630 387.3	4 956.5	-	-	-	-	-	-	627 127	627 127	656.9	**579 934**	623 430.9	99.0
	4	-	-	-	-	-	-	980 763	996 916.7	1 843.7	**979 512**	995 338.7	5 049.9	-	-	-	-	-	-	1 126 042	1 126 042	1 049.7	1 028 505	1 098 335.0	162.9
	8	-	-	-	-	-	-	1 369 562	1 383 706.2	2 053.2	**1 365 712**	1 379 907.1	5 226.3	-	-	-	-	-	-	1 484 155	1 484 155	1 313.3	1 492 866	1 552 394.9	207.9
	16	-	-	-	-	-	-	1 792 255	1 822 434.2	2 619.0	**1 786 603**	1 815 823.4	5 805.4	-	-	-	-	-	-	1 931 414	1 931 414	1 513.8	1 934 580	1 963 588.6	242.5
	32	-	-	-	-	-	-	2 201 402	2 228 298.4	3 362.3	**2 194 574**	2 219 372.7	6 577.8	-	-	-	-	-	-	2 359 214	2 359 214	1 661.0	2 320 914	2 357 586.6	269.1
	64	-	-	-	-	-	-	2 582 763	2 605 741.5	4 200.2	**2 573 308**	2 595 415.7	7 444.6	-	-	-	-	-	-	2 693 068	2 693 068	1 750.7	2 714 084	2 756 581.6	291.0
	128	-	-	-	-	-	-	2 922 845	2 938 557.5	5 244.8	**2 912 304**	2 926 775.7	8 411.7	-	-	-	-	-	-	3 021 405	3 021 405	1 825.8	3 067 141	3 083 256.1	306.9
nlpkkt160	2	104 429	106 067.1	9 082.23	**103 680**	104 995.1	43 027.78	128 575	133 762.6	2 858.16	124 559	130 257.9	8 587.27	129 196	131 062.2	15 365.75	**103 680**	**103 680.0**	14 184.25	**103 680**	**103 680**	241.79	107 441	116 562.7	83.09
	4	213 973	220 365.8	16 183.60	214 334	222 224.7	79 019.67	259 707	268 212.5	2 883.26	252 630	261 784.2	8 693.45	259 417	263 255.4	15 427.79	**206 072**	206 342.6	25 683.03	211 144	211 144	470.25	219 501	228 302.0	151.45
	8	322 856	329 548.9	23 544.55	320 994	324 373.0	152 186.00	391 612	396 803.5	2 913.92	381 239	386 972.5	8 797.02	387 817	391 016.8	15 514.50	**307 192**	307 691.8	35 485.60	325 600	325 600	897.19	332 655	347 803.2	215.29
	16	521 155	528 080.4	37 285.94	518 893	529 104.4	141 926.80	615 205	627 955.8	2 967.54	599 618	612 278.9	8 846.43	609 817	619 657.0	15 645.37	**509 274**	509 929.8	43 027.35	517 056	517 056	1 108.15	542 561	563 872.8	275.02
	32	734 161	743 979.2	53 097.82	730 191	741 399.0	210 242.56	856 823	866 654.9	3 026.94	834 321	844 164.5	9 006.93	849 811	855 207.3	15 964.52	**708 178**	709 038.0	50 371.74	719 978	719 978	1 297.96	766 241	791 548.0	329.29
	64	957 559	965 291.9	71 637.31	958 350	963 160.75	216 510.75	1 112 493	1 121 800.4	3 122.96	1 081 252	1 090 925.1	9 134.81	1 106 395	1 112 384.9	16 248.75	**901 676**	903 211.9	55 197.10	927 360	927 360	1 496.62	972 314	1 025 149.3	379.14
	128	1 321 426	1 327 462.1	103 657.70	1 317 083	1 325 698.6	195 255.28	1 508 972	1 513 050.5	3 325.30	1 466 125	1 470 326.7	9 417.42	1 481 113	1 486 507.7	16 875.58	**1 284 197**	1 286 356.7	59 681.49	1 301 624	1 301 624	1 723.02	1 375 724	1 397 064.5	425.90

H	k	KaHyPar-Strong min	KaHyPar-Strong avg	KaHyPar-Strong T[s]	KaHyPar-StrongV min	KaHyPar-StrongV avg	KaHyPar-StrongV T[s]	KaHyPar-Fast min	KaHyPar-Fast avg	KaHyPar-Fast T[S]	KaHyPar-FastV min	KaHyPar-FastV avg	KaHyPar-FastV T[s]	hMetis-K min	hMetis-K avg	hMetis-K T[s]	hMetis-R min	hMetis-R avg	hMetis-R T[s]	PaToH-Q min	PaToH-Q avg	PaToH-Q T[s]	PaToH-D min	PaToH-D avg	PaToH-D T[s]
thermal2	2	931	937.7	65.40	**928**	934.9	214.96	1 031	1 082.3	61.19	1 017	1 057.6	183.59	973	998.9	236.38	964	978.1	258.10	990	990	9.29	980	997.8	2.27
	4	2 798	2 807.8	76.10	**2 787**	2 798.8	367.54	3 101	3 198.1	61.33	3 038	3 123.6	183.85	2 984	3 022.9	257.54	2 891	2 915.1	488.52	3 035	3 035	17.43	2 989	3 039.9	4.14
	8	6 549	6 571.0	97.88	**6 526**	6 547.2	607.21	7 338	7 404.8	61.67	7 164	7 237.4	184.49	7 046	7 091.9	261.67	6 799	6 822.4	671.76	7 037	7 037	25.09	7 080	7 302.3	5.80
	16	10 883	10 930.7	124.69	**10 826**	10 891.3	868.21	12 206	12 333.6	62.32	11 931	12 036.0	185.50	11 800	11 891.0	266.80	11 485	11 571.2	808.48	11 826	11 826	32.29	12 076	12 354.9	7.29
	32	18 099	18 203.1	170.91	**18 014**	18 123.7	1 532.35	20 175	20 327.5	64.48	19 712	19 863.9	187.96	19 436	19 537.5	273.79	19 116	19 195.3	918.85	19 559	19 559	38.47	20 014	20 253.0	8.70
	64	26 991	27 120.5	234.94	**26 886**	27 003.5	2 057.76	29 859	30 057.4	69.99	29 227	29 391.4	193.29	29 036	29 289.5	287.47	28 494	28 591.2	1 028.03	29 339	29 339	43.19	30 298	30 678.7	10.10
	128	40 771	40 962.4	345.00	**40 560**	40 774.8	3 087.87	45 058	45 181.8	80.37	44 114	44 237.5	204.07	43 546	43 780.7	314.43	43 097	43 219.2	1 091.11	44 048	44 048	52.66	45 139	45 511.4	11.51
bcsstk29	2	**360**	363.0	1.03	**360**	360.6	2.29	372	400.8	0.41	366	387.6	1.00	**360**	363.6	2.70	**360**	**360.0**	3.17	**360**	**360**	0.42	372	387.0	0.12
	4	1 080	1 092.6	2.53	**1 080**	1 086.0	6.44	1 170	1 206.0	0.57	1 152	1 180.8	1.27	1 092	1 104.0	3.17	1 086	1 088.4	6.84	1 158	1 158	0.58	1 146	1 198.2	0.22
	8	2 190	2 227.8	5.05	**2 184**	2 212.8	12.51	2 352	2 396.4	0.99	2 340	2 386.8	1.58	2 298	2 351.4	4.68	2 220	2 237.4	11.11	2 428	2 428	1.11	2 469	2 536.3	0.33
	16	3 582	3 643.4	9.16	**3 540**	3 626.5	22.71	3 718	3 781.1	1.98	3 718	3 781.1	2.28	3 828	3 898.4	7.87	3 726	3 774.6	15.42	3 870	3 870	1.41	3 920	4 003.1	0.42
	32	5 202	5 283.8	14.52	**5 196**	5 266.6	38.77	5 321	5 360.3	4.80	5 315	5 356.1	5.34	5 427	5 520.6	14.21	5 296	5 393.8	19.68	5 525	5 525	1.69	5 668	5 808.1	0.50
	64	6 998	7 106.8	22.59	**6 992**	7 104.8	34.42	7 033	7 126.4	14.97	7 032	7 124.6	15.49	7 254	7 314.3	29.18	7 115	7 242.0	23.36	7 731	7 731	1.83	7 758	7 890.2	0.56
	128	10 782	10 823.3	25.94	10 782	10 823.3	25.74	10 782	10 823.3	25.91	10 782	10 823.3	26.20	9 979	10 088.8	38.06	10 764	10 847.6	25.66	**9 865**	**9 865**	2.06	10 004	10 058.6	0.61
bcsstk30	2	**527**	529.3	2.97	**527**	529.3	5.82	**527**	572.0	1.71	**527**	569.5	3.21	535	544.5	5.73	**527**	552.8	7.59	578	578	1.28	528	577.9	0.56
	4	1 482	1 521.2	6.43	**1 481**	1 494.2	18.27	1 559	1 610.0	2.04	1 517	1 588.7	4.94	1 488	1 544.7	7.03	1 524	1 598.1	18.56	1 572	1 572	2.46	1 576	1 657.5	1.08
	8	3 103	3 156.7	13.40	**3 103**	3 140.2	31.07	3 234	3 378.0	2.94	3 159	3 298.9	6.47	3 206	3 323.6	9.92	3 264	3 336.1	27.85	3 394	3 394	4.72	3 402	3 672.7	1.59
	16	6 555	6 658.0	56.71	**6 522**	6 610.3	199.96	6 726	6 805.7	5.42	6 684	6 753.6	9.91	6 884	7 207.3	14.39	6 844	7 069.3	35.28	6 817	6 817	5.66	7 120	7 601.5	2.04
	32	10 212	10 368.0	90.30	**10 098**	10 282.5	409.05	10 303	10 475.0	10.59	10 260	10 399.3	15.72	11 020	11 370.8	27.42	10 957	11 161.9	42.62	10 942	10 942	6.81	11 001	11 415.9	2.43
	64	14 537	14 723.9	123.11	**14 465**	14 661.5	556.62	14 513	14 701.0	22.01	14 467	14 676.2	28.87	15 927	16 072.5	49.40	15 219	15 488.0	49.36	14 859	14 859	7.29	15 461	15 832.5	2.70
	128	19 789	20 048.6	129.07	19 789	20 045.6	272.81	19 758	20 009.2	35.92	19 750	19 971.7	41.04	21 288	21 493.1	74.78	20 657	20 921.0	53.13	19 633	**19 633**	8.11	20 146	20 339.0	2.90
bcsstk31	2	**664**	684.1	2.78	**664**	678.2	6.36	691	743.6	0.90	677	728.7	2.36	674	695.6	8.44	667	672.7	9.85	678	678	0.69	665	744.7	0.22
	4	1 628	1 684.8	5.84	**1 628**	1 678.3	12.44	1 783	1 841.1	1.20	1 685	1 786.1	2.86	1 660	1 711.8	11.37	1 687	1 720.0	19.21	1 975	1 975	1.69	1 725	1 953.3	0.42
	8	3 222	3 345.3	13.39	**3 192**	3 329.6	40.99	3 431	3 601.4	2.08	3 388	3 545.0	3.71	3 361	3 425.0	13.05	3 244	3 391.7	29.20	3 494	3 494	2.37	3 602	3 867.4	0.62
	16	5 542	5 717.1	24.08	**5 470**	5 665.2	83.45	6 034	6 136.7	4.32	5 982	6 042.4	5.91	5 875	5 961.4	18.30	5 832	5 922.8	38.89	6 171	6 171	2.64	6 402	6 702.5	0.82
	32	8 841	8 927.9	43.57	**8 693**	8 864.4	163.41	9 355	9 427.5	9.39	9 224	9 338.4	10.95	9 370	9 701.3	29.68	9 081	9 299.1	48.96	9 382	9 382	3.87	9 681	10 070.8	1.00
	64	12 913	13 056.8	69.88	**12 829**	12 946.5	292.55	13 445	13 535.2	20.00	13 365	13 464.5	21.52	13 889	14 021.9	55.40	13 101	13 325.2	59.77	13 846	13 846	4.57	14 088	14 578.7	1.17
	128	17 975	18 083.4	84.76	**17 805**	17 939.6	393.91	18 171	18 316.3	38.40	18 144	18 282.6	39.58	19 180	19 311.3	92.71	18 150	18 260.6	69.10	18 828	18 828	5.41	19 438	19 548.3	1.33
bcsstk32	2	**831**	831.6	3.61	**831**	831.6	7.20	850	923.4	1.22	844	905.1	3.29	986	1 029.1	9.77	958	1 036.9	14.18	918	918	1.40	966	1 032.9	0.39
	4	1 589	1 744.1	6.75	**1 588**	1 738.4	16.30	1 725	1 899.6	1.42	1 675	1 850.2	3.73	1 693	1 764.8	11.84	2 118	2 207.2	27.52	1 696	1 696	1.82	1 707	2 134.8	0.74
	8	3 618	3 733.4	13.85	**3 568**	3 716.2	42.34	3 925	4 045.8	2.00	3 829	3 972.0	4.32	3 752	3 979.8	14.75	3 792	4 089.4	40.24	4 125	4 125	2.77	3 804	4 276.6	1.08
	16	6 205	6 351.1	24.75	**6 193**	6 331.5	74.18	6 581	6 778.0	3.52	6 506	6 700.2	5.84	6 536	6 863.3	18.31	6 518	6 711.1	53.60	6 801	6 801	4.02	6 875	7 181.9	1.41
	32	9 839	9 960.1	39.79	**9 789**	9 905.1	140.27	10 302	10 454.5	6.93	10 254	10 397.8	9.21	10 562	10 848.5	27.56	10 207	10 504.7	67.01	10 551	10 551	4.71	10 703	11 113.0	1.72
	64	13 948	14 266.1	61.10	**13 764**	14 149.5	322.70	14 683	14 933.4	15.08	14 601	14 883.6	17.33	15 521	15 812.9	44.46	14 940	15 135.5	77.41	15 273	15 273	6.02	15 809	16 073.9	2.00
	128	19 818	20 008.3	83.35	**19 670**	19 852.2	447.77	20 217	20 348.5	30.34	20 159	20 295.2	32.27	21 788	21 972.9	76.66	20 343	20 853.1	91.15	21 315	21 315	7.41	22 008	22 326.3	2.27
memplus	2	2 908	2 916.5	17.08	2 902	2 911.6	50.30	2 948	2 973.1	0.56	2 948	2 967.5	1.28	2 696	2 710.8	1.63	**2 465**	2 513.7	2.23	2 983	2 983	0.22	2 922	3 051.2	0.06
	4	4 217	4 904.1	26.07	4 065	4 859.6	136.76	4 741	5 040.5	0.87	4 741	5 028.0	1.70	4 133	4 147.1	2.25	**3 974**	4 069.5	3.20	4 422	4 422	0.26	4 354	4 488.2	0.07
	8	5 541	5 854.1	40.21	**4 983**	5 675.0	258.62	5 659	5 919.5	1.00	5 656	5 887.3	1.60	5 023	5 057.5	3.22	5 061	5 118.6	4.12	5 101	5 101	0.35	5 132	5 230.8	0.08
	16	6 387	6 550.7	63.39	6 090	6 283.6	566.70	6 236	6 407.7	1.59	6 228	6 371.4	2.24	**5 703**	5 736.6	5.52	5 719	5 774.6	4.94	5 756	5 756	0.41	5 733	5 900.0	0.09
	32	6 518	6 572.9	89.58	6 495	6 554.4	449.99	6 546	6 590.3	2.42	6 546	6 586.2	2.95	6 389	6 419.5	12.38	**6 273**	6 290.2	5.87	6 359	6 359	0.45	6 400	6 475.5	0.09
	64	6 961	7 009.6	111.88	6 958	7 004.2	403.02	6 941	6 993.7	4.32	6 939	6 991.8	4.56	6 962	7 020.2	38.08	**6 701**	6 782.5	7.08	6 914	6 914	0.50	6 880	6 940.0	0.11
	128	7 298	7 346.4	8.79	7 298	7 346.4	8.79	7 298	7 346.4	8.78	7 298	7 346.4	8.83	7 547	7 570.1	68.98	7 317	7 364.1	8.72	**7 288**	**7 288**	0.56	**7 288**	7 333.3	0.12
vibrobox	2	2 489	2 498.8	25.80	2 485	2 494.7	78.63	2 511	2 583.4	1.08	2 510	2 566.6	1.91	**1 990**	**1 990.0**	4.81	**1 990**	**1 990.0**	5.51	**1 990**	**1 990**	0.51	**1 990**	2 213.3	0.13
	4	4 091	4 204.4	64.71	3 913	4 108.0	264.34	4 275	4 399.8	2.02	4 102	4 352.6	3.11	3 796	3 798.5	7.79	**3 598**	3 604.0	9.73	3 660	3 660	0.73	3 642	3 945.5	0.18
	8	4 747	4 855.8	91.26	**4 722**	4 819.0	305.06	4 836	5 012.4	3.53	4 822	4 987.9	4.53	5 932	6 018.6	13.48	4 867	4 913.0	13.32	4 897	4 897	0.84	5 027	5 248.1	0.22
	16	5 152	5 418.4	125.71	**5 094**	5 332.8	625.73	5 268	5 574.4	6.24	5 213	5 531.1	7.10	7 293	7 381.9	22.74	6 018	6 174.8	16.89	6 284	6 284	1.06	6 477	6 557.5	0.25
	32	6 818	6 910.8	176.94	**6 722**	6 864.0	946.83	6 866	6 949.0	11.33	6 858	6 939.9	12.12	8 440	8 492.9	38.58	6 905	6 989.5	20.24	7 172	7 172	1.26	7 242	7 342.9	0.28
	64	7 748	7 820.8	94.19	**7 745**	7 816.3	284.09	7 741	7 808.6	20.28	7 740	7 806.8	20.60	9 635	9 683.8	59.93	7 831	7 878.7	23.64	7 971	7 971	1.25	8 023	8 135.4	0.31
	128	8 751	8 811.5	27.15	8 751	8 811.5	27.10	8 751	8 811.5	27.14	8 751	8 811.5	27.13	10 344	10 389.5	79.75	**8 722**	8 793.8	27.26	9 011	9 011	1.59	8 957	9 054.8	0.33
finan512	2	**146**	**146.0**	1.96	**146**	**146.0**	3.63	147	147.6	1.06	146	147.5	2.17	**146**	146.8	7.65	**146**	146.3	9.37	148	148	0.50	147	147.9	0.15
	4	**292**	**292.0**	3.17	**292**	**292.0**	5.53	294	294.9	1.34	294	294.7	2.44	293	294.3	8.14	**292**	293.2	18.01	296	296	0.93	295	303.2	0.27
	8	**584**	591.3	5.72	**584**	591.3	9.34	586	596.2	2.18	586	596.0	3.25	589	590.0	9.95	**584**	585.1	27.23	592	592	1.53	591	621.1	0.41
	16	**1 168**	1 168.0	11.70	**1 168**	1 168.0	17.75	1 169	1 176.3	4.88	1 169	1 176.1	5.92	1 180	1 180.9	14.34	**1 168**	1 172.1	37.34	1 183	1 183	2.22	1 184	1 264.8	0.55
	32	**2 336**	**2 336.0**	27.25	**2 336**	**2 336.0**	39.80	2 350	2 362.2	11.36	2 350	2 361.1	12.51	2 357	2 360.7	29.79	**2 336**	2 342.4	50.54	2 366	2 366	2.95	2 366	2 366.8	0.72
	64	9 037	9 122.1	74.26	9 015	9 080.7	206.08	9 348	9 541.5	24.51	9 214	9 368.0	26.38	9 063	9 077.0	61.78	**8 915**	8 946.5	67.39	9 142	9 142	4.04	9 605	10 188.2	0.92
	128	18 219	18 440.1	112.22	18 069	18 219.0	580.78	18 836	19 148.6	44.58	18 561	18 812.8	46.26	18 136	18 148.3	110.18	**17 792**	17 921.2	86.43	19 319	19 319	5.13	20 962	21 736.5	1.10