HAL
open science

# STS-k: A Multilevel Sparse Triangular Solution Scheme for NUMA Multicores

Humayun Kabir, Joshua Booth, Guillaume Aupy, Anne Benoit, Yves Robert, Padma Raghavan

## To cite this version:

Humayun Kabir, Joshua Booth, Guillaume Aupy, Anne Benoit, Yves Robert, et al.. STS-k: A Multilevel Sparse Triangular Solution Scheme for NUMA Multicores. [Research Report] RR-8763, Penn State University; ENS Lyon; INRIA. 2015. hal-01183904

**HAL Id: hal-01183904**

**https://inria.hal.science/hal-01183904v1**

Submitted on 11 Aug 2015

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# STS-k: A Multilevel Sparse Triangular Solution Scheme for NUMA Multicores

**Humayun Kabir, Joshua Booth, Guillaume Aupy, Anne Benoit, Yves Robert, Padma Raghavan**

**RESEARCH REPORT**
**N° 8763**
August 2015
Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--8763--FR+ENG

# STS-k: A Multilevel Sparse Triangular Solution Scheme for NUMA Multicores

Humayun Kabir*, Joshua Booth†, Guillaume Aupy‡, Anne Benoit‡, Yves Robert‡§¶, Padma Raghavan*

Project-Team ROMA

Research Report n° 8763 — August 2015 — 29 pages

**Abstract:** We consider techniques to improve the performance of parallel sparse triangular solution on non-uniform memory architecture multicores by extending earlier coloring and level set schemes for single-core multiprocessors. We develop STS-k, where $k$ represents a small number of transformations for latency reduction from increased spatial and temporal locality of data accesses. We propose a graph model of data reuse to inform the development of STS-k and to prove that computing an optimal cost schedule is NP-complete. We observe significant speed-ups with STS-3 on 32-core Intel Westmere-EX and 24-core AMD 'MagnyCours' processors. Incremental gains solely from the 3-level transformations in STS-3 for a fixed ordering, correspond to reductions in execution times by factors of 1.4 (Intel) and 1.5 (AMD) for level sets and 2 (Intel) and 2.2 (AMD) for coloring. On average, execution times are reduced by a factor of 6 (Intel) and 4 (AMD) for STS-3 with coloring compared to a reference implementation using level sets.

**Key-words:** Linear algebra; distributed computing; NUMA architecture.

* Department of Computer Science & Engineering, Pennsylvania State University, USA
† Center for Computer Research, Sandia National Laboratories, USA
‡ LIP, École Normale Supérieure de Lyon, INRIA, France
§ Institut Universitaire de France
¶ University of Tennessee Knoxville, USA

**RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES**

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

# STS-k: Solveur multi-niveaux de systèmes triangulaires creux pour architectures NUMA

**Résumé :** Nous montrons comment améliorer les performances des solveurs parallèles pour les systèmes linéaires triangulaires creux sur des architectures multi-cœurs, en étendant les techniques *coloring* et *level set* utilisées sur des clusters de processeurs à un seul cœur. Nous présentons STS-k, où le paramètre $k$ représente le nombre de transformations pour réduire la latence. Nous proposons une modélisation de la ré-utilisation des données, et nous établissons la NP-complétude du problème d'ordonnancement associé.

Les facteurs d'accélération obtenus avec STS-3 observés sont notables, que ce soit sur des processeurs Intel Westmere-EX 32-core ou sur des AMD 'Magny-Cours' 24-core. Les temps d'exécution sont réduits en moyenne d'un facteur 6 (intel) et 4 (AMD) pour STS-3 utilisant *coloring* comparé à une implémentation de référence qui utilise *level-set*.

**Mots-clés :** Algèbre linéaire ; calcul distribué ; architecture NUMA.

# 1 Introduction

Sparse triangular solutions are required in many scientific applications, and particularly when sparse linear systems are solved using a method such as preconditioned conjugate gradient. Parallel implementations of sparse triangular solution tend to perform poorly for several reasons. The dependencies arising from the sparsity structure can reduce the number of tasks that can be computed in parallel. There are often significant overheads related to synchronization after the completion of each set of independent tasks. Furthermore, higher data access latencies can degrade the efficiency of an implementation especially on multicores with large core counts and non-uniform memory architectures (NUMA). These factors present new opportunities for speeding-up sparse triangular solution on NUMA multicores [12]. An added benefit is that any performance gains that can be achieved at a multicore node will also increase efficiencies when larger problems are solved across multiple multicore nodes of a high performance computing cluster.

We consider sparse triangular solutions with a lower triangular coefficient matrix $L$, i.e., solving for $x$ in the system $L \times x = b$. We provide a new multilevel method that we call STS-k, where $k$ is a small integer that reflects the number of transformations. These transformations lead to $k$-levels of sub-structuring of the data with large independent sets or packs of tasks that reuse components of the solution vector. These packs can be scheduled on a NUMA multicore to promote reuse through temporal locality for enhanced levels of performance from reduced latencies of data accesses.

The remainder of this paper is organized as follows. Section 2 contains a brief overview of parallel sparse triangular solution and prior related research. Section 3 contains our main contributions. These include a Data Affinity and Reuse (DAR) task graph model to study the complexity of optimal scheduling schemes, to prove that computing the schedule of minimum time is NP-complete, and to formulate heuristics that inform the development of STS-k. Section 4 concerns an in-depth empirical evaluation of STS-k, with $k$=3 on 1-32 cores of an Intel Westmere-EX processor and on 1-24 cores of an AMD 'MagnyCours' processor. Section 5 contains concluding remarks including directions for future research.

# 2 Background and Related Work

In this section, we provide a brief overview of sparse triangular solution and review of related work.

**Sparse Triangular Solution.** Sparse triangular solution concerns a linear system of equations of the form $L \times x = b$, where $L$ is a sparse lower triangular matrix and $b$ a vector. Components of the solution vector $x$ are computed using elements of $L$ and $b$. Observe that if the matrix $L$ were dense, then triangular solution would be inherently sequential in that the solution for component $x_i$ would depend on all previous components, i.e., $x_1, x_2, \ldots, x_{i-1}$, and hence the

components have to be computed in order from the first to the last one. The sparsity of $L$, i.e., the presence of zeroes, removes this constraint although there will be dependencies among groups of the solution vector components. These dependencies need to be managed for effective parallelization and to promote data reuse to reduce latencies of data accesses on NUMA multicores.

**Related Research.** There are two alternative techniques, one based on coloring and the other on level sets, to compute orderings that expose parallelism in sparse triangular solution.

The earliest scheme, based on graph coloring, was proposed by Schreiber and Tang [9] in the eighties to extract parallelism in sparse triangular solution for efficient implementations on multiprocessors. Consider the graph of the matrix $A = L + L^T$, $G = (V, E)$ where each row/column in $A$ corresponds to a vertex in $G$ and an edge exists between vertex $i$ and $j$ if and only if $A_{ij} \neq 0$. Now $G$ is colored, followed by a reordering of the matrix $L$ in which rows/columns in each color are numbered contiguously, in some order of the colors. This method is very effective in identifying large independent sets of tasks that can be processed in parallel to calculate corresponding components of the solution vector.

In the nineties, a level set reordering was proposed by Saltz [8] as a scheme to enable parallel sparse triangular solution on multiprocessors. In this approach, a variant of breadth-first search is performed on the graph of $A = L + L^T$ to identify a succession of vector solution components that can be computed concurrently. Each level identified in the search corresponds to a set of calculations that can be performed in parallel after the previous levels have been completed.

A *parallel step* in sparse triangular solution corresponds to solving unknowns within the same color or same level, concurrently. These parallel steps must be performed one after the other because they depend on unknowns that were computed in a preceding step, i.e., at a preceding color or level. Consequently, their total number, which is equal to the number of colors or levels, and the average number of solution components per step provide a measure of available parallelism. The total number of these steps is also a measure of the overheads of synchronization that must occur within an implementation of parallel sparse triangular solution.

More recently, Naumov [6] developed a sparse triangular solution scheme for Nvidia GPUs using the level set ordering to extract parallelism. We utilized the fact that graph coloring is very effective in extracting large independent sets of calculations to develop an alternate scheme for sparse triangular solution on GPUs [11]. This scheme resulted in very large factors of performance gains relative to the Nvidia implementation based on level set ordering [6].

Wolf et al. [12] considered several factors influencing the performance of sparse triangular solution on multicores to conclude that the overheads of synchronization between parallel steps have the greatest effect on the performance of sparse triangular solution. Earlier, we had proposed a multilevel compressed sparse row sub-structuring (CSR-k) as an alternative to the traditional compressed sparse row format for sparse matrix calculations on NUMA multicores [4]. This sub-structuring of the sparse matrix seeks to mimic the multiple levels of caches present in NUMA multicores to increase locality of data

accesses. Our evaluation for parallel sparse matrix vector multiplication indicated significant performance benefits largely as a consequence of higher cache hit rates and an effective reduction in data access latencies.

In this paper, we adapt CSR-k, coloring and level-set approaches and develop a scheduling framework for efficient parallel sparse triangular solution on NUMA multicores.

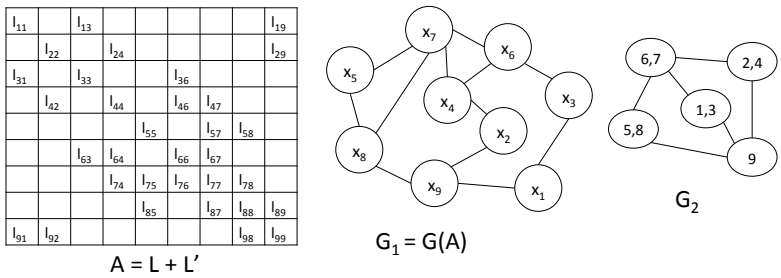


Figure 1: $A = L + L^T$ (left) and its graph $G_1$ (middle) transformed into $G_2$ (right) with super-rows through coarsening. A vertex of $G_2$ is formed by collapsing two connected vertices of $G_1$.

# 3 STS-k: A Multilevel Sparse Triangular Solution Scheme for NUMA Multicores

In this section, we develop STS-k, a multilevel scheme for sparse triangular solution to achieve high performance on NUMA multicores. The key idea is to restructure the traditional scheme to exploit the memory hierarchy of NUMA multicores that affect data access latencies.

We develop STS-k as a specific composition of key techniques. We first consider techniques to increase: (i) spatial locality of memory accesses by grouping rows into super-rows; and (ii) the size of independent sets of super-rows that can be processed in parallel. Next, we develop a NUMA-aware task graph model of calculations in an independent set, or *pack*, and specify a scheduling problem, which we show to be NP-complete. Then, we present a special case of the scheduling problem and its heuristic solution. These results inform a further level of sub-structuring to promote data reuse through temporal locality of shared solution components among tasks in a pack to yield STS-k. We expect that STS-k will permit high levels of parallelism and spatial and temporal locality of data accesses to significantly speed-up sparse triangular solution on NUMA multicores.

### 3.1 Super-rows for Increasing Spatial Locality in Data Accesses

In our previous work [4], we introduced CSR-k, a multi-level compressed sparse row format for efficient sparse matrix-vector multiplication, SpMV, on multicore processors. In this scheme, $k$ is an integer greater than 1 that represents the number of well-differentiated levels in the memory hierarchy of a multicore processor and it is used to repeatedly ($k-1$ times) group rows of the matrix into a format that increased the spatial locality of access during SpMV. We now adapt CSR-k to improve spatial locality in accesses to $L$ and $x$ in sparse triangular solution.

CSR-k uses the graph form of a symmetric sparse matrix; in sparse triangular solution we have only a sparse lower triangular matrix $L$ but we can construct an equivalent symmetric matrix $A = L + L^T$ and consider its undirected graph $G_1$ (see Figure 1 for an example). Consider a number of rows of $A$ that have in common one or more columns with nonzeroes in them, or equivalently, their vertex counterparts in $G_1$ that are connected by many edges. The latter can be agglomerated to form a super-vertex with the corresponding rows in $A$ forming a super-row. This agglomeration can be formalized using graph coarsening or grouping together continuous rows of $A$, if it is ordered using band-reducing ordering, such as reverse Cuthill-McKee [1] (more details can be found in [4]). The coarsened graph corresponding to this grouping, whose vertices now represent super-rows, is denoted by $G_2$. For example, in Figure 1, each *super-row* consists of two rows of $A$ that share nonzeroes in a column. In CSR-k this process is repeated $k-1$ times to get super-rows, super-super-rows and so on. Further, coarsening is performed in such a way to seek super-rows with equal numbers of nonzeroes to enable equal work tasks.

When CSR-k is developed for performing parallel SpMV, observe that there are no dependencies among tasks that operate on super-rows. Instead, there are opportunities to exploit spatial locality in accessing the matrix, the vector and potentially promoting reuse in the vector [4]. Consequently, $k$ could be selected and combined with certain data reordering and restructuring techniques purely to promote locality at various levels of the NUMA cache hierarchy [4]. However, parallelism in sparse triangular solution is markedly more limited and complex. Consequently, we recommend keeping $k$ to a very small value of 1 or 2 to increase spatial locality to a certain extent while retaining a large fraction of the fine-grained connectivity structure to extract parallelism before additional locality enhancing techniques are applied.

### 3.2 Identifying Large Packs or Sets of Independent Super-rows for Parallel Processing

Traditionally, methods such as level-set [6, 8] and coloring [9, 11] have been applied to $G_1$, the graph of $A_1 = L + L^T$ to identify independent sets of solution vector components so that the associated tasks can be performed in parallel. The number and sizes of such independent sets matter with larger and fewer

sets being more desirable. This is because larger sets represent a greater degree of available parallelism. Further, there are dependencies between these independent sets in that they have to be computed one after the other in order to ensure that components computed in earlier sets are available for use in later sets; a smaller number of sets leads to fewer synchronization points during sparse triangular solution. As discussed in Section 2, coloring is generally superior to level-set in regard to these factors.

To identify independent set of super-rows and the subsets of their solution vector components that can be computed in parallel, either level-set or coloring can be applied to $G_2$. We expect that coloring will be more effective than level-set in producing fewer and larger independent sets or *packs*. Further, we expect that level-set may actually do better when applied to $G_2$ as opposed to $G_1$ because $G_2$ has fewer vertices leading to a fewer levels in level-set and thus fewer independent sets representing larger amounts of parallel work. To further accentuate this effect, we propose that level-set be applied starting with a vertex of largest degree. Assume that some $n_3$ packs are obtained and that they are denoted by $P_k, k \in 1, \ldots, n_3$. Typically, tasks related to solving for unknowns in $P_k$ will depend on solution components from all previous packs $P_1, \ldots, P_{k-1}$. To increase the level of reuse of solution components from earlier independent sets, we propose ordering the independent sets in increasing order of their sizes, i.e., find a permutation $\pi$ such that $|P_{\pi(1)}| \leq |P_{\pi(2)}| \leq \cdots \leq |P_{\pi(n_3)}|$. Figure 2 shows $G_2$ from the earlier example organized into independent sets or packs after coloring.

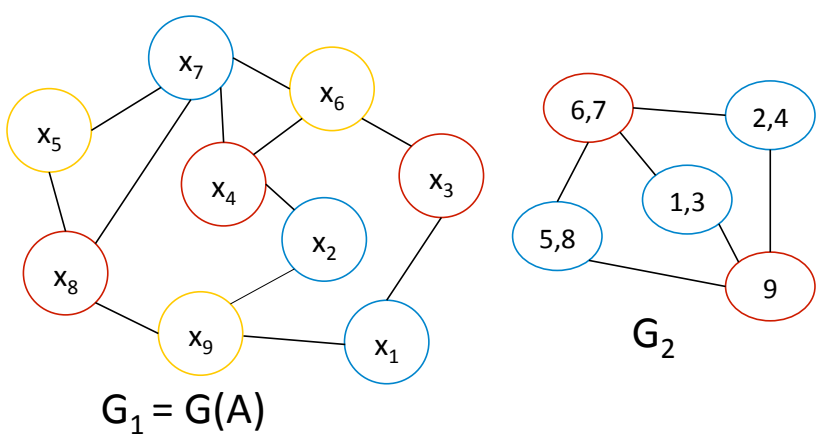


Figure 2: Independent sets or packs of rows obtained after coloring of $G_1$ (left) and packs of super-rows after coloring of $G_2$ of the example shown earlier in Figure 1. Observe the differences including 3 colors for $G_1$ versus 2 in $G_2$.

### 3.3 NUMA-aware Scheduling of Packs for Utilizing Temporal Locality of Data Accesses

Once packs and their ordering have been determined, their constituent tasks can be processed in order from pack $P_1$ through to $P_{n_3}$. We now consider scheduling tasks in a pack in order to utilize the sharing of specific solution components from a previous pack among tasks in the current pack.

Without loss of generality, assume that a task corresponds to a super-row and its associated solution components. Such a task would require arithmetic operations proportional to the number of nonzeroes in the associated super-row. By construction, each super-row was selected to have approximately the same number of non-zeroes and so the number of arithmetic operations per task could be assumed to be equal and of a certain value $e$. However, actual execution times would depend on the cost of data accesses. One way to reduce latencies is by promoting reuse of solution components from proximal caches in the NUMA sense. We now seek to formalize these costs so we may study how tasks within a pack may be scheduled in a NUMA-aware manner.

We formulate a *data affinity and reuse*, i.e., a DAR graph of pack $k$, $\mathrm{DAR}(k) = (V_k, E_k)$, where $V_k = \{t_m \in P_k\}$, the set of tasks in $P_k$. We denote the unknowns in a task $t_l$ by $x_l$ and the set of unknowns on which the solution of $x_l$ depends upon is denoted by $DX_l$. Now, there is an edge between tasks $t_l$ and $t_m$, i.e., $(t_l, t_m) \in E_k$, if $DX_l \cap DX_m \neq \emptyset$.

A DAR of a pack is shown in Figure 3. We want a schedule of the tasks in a pack, i.e., tasks to cores mapping is sought in order to balance the load across cores and to minimize data access costs based on a latency model that concern NUMA-distances between cores. First, we show that this scheduling problem is NP-complete even on a Uniform Memory Access (UMA) architecture. After that we propose a heuristic algorithm for this assignment problem which is useful for a particular kind of DAR.

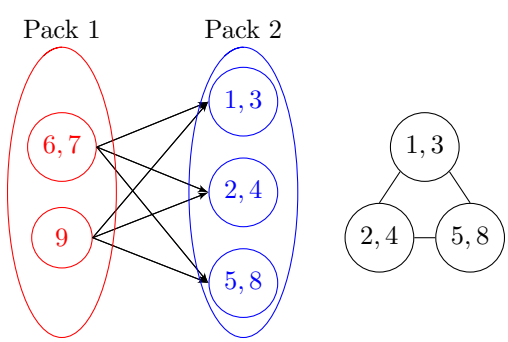


Figure 3: The construction of packs (left) and data affinity and reuse (DAR) graph of tasks of the second pack of $G_2$ (right) based on the example shown in Figure 2. Each task corresponds to a super-row and two tasks are connected if they need elements of $x$ computed in the first pack.

To assess the complexity, we start by simplifying the cache-model of the processor as much as possible while retaining the combinatorial nature of the problem. It turns out that we can consider a single level of cache with unlimited capacity. Roughly speaking, this would be similar to only considering the (very large) $L3$ cache.

**Definition 1** (One-level platform)**.** *We suppose that we have $q$ processors that share a storage location of infinite volume (called memory). There are $q$ caches $C_1, \ldots, C_q$ such that:*

- *Processor $j$ can only read data stored in cache $C_j$ at a cost $r$.*
- *The cost to copy some unit of data from memory to any cache is $w$.*
- *Each cache has an infinite volume.*

**Definition 2** (In-Pack)**.** In-pack affinity-aware assignment problem*: We have a set of tasks $(t_1, \ldots, t_n)$ that we want to schedule on $q$ processors. We assume that the tasks have equal execution time $e$. Let $I_i$ be the set of data inputs necessary for the execution of $t_i$. The* In-Pack *problem is to schedule the $n$ tasks on $q$ processors while minimizing execution time.*

In order to schedule the $n$ tasks on the $q$ processors, we need to partition the tasks onto these processors. Let $\pi$ be a schedule: task $t_i$ is mapped onto processor $\pi(t_i)$. For $1 \leq j \leq q$, let $V_j = \{i \mid \pi(t_i) = j\}$ denote the indices of tasks mapped onto processor $j$. The execution time of processor $j$ is the sum of:

- the cost of transferring required data inputs to its cache:

$$w \cdot |\cup_{i \in V_j} I_i|;$$

- the time to execute the tasks:

$$e \cdot |V_j|;$$

- the time to perform the required number of reads:

$$r \cdot \sum_{i \in V_j} |I_i|.$$

  Note that we have the sum of the cardinals $\sum |I_i|$ rather than the cardinal of their union $|\bigcup I_i|$ (as for the copy from memory to cache), because the input of the ALU buffers must be reloaded for each task.

Finally, the execution time of a schedule is:

$$\max_{1 \leq j \leq q} \left( w \cdot \Big| \bigcup_{i \in V_j} I_i \Big| + e \cdot |V_j| + r \cdot \sum_{i \in V_j} |I_i| \right). \tag{1}$$

The decision problem associated to In-Pack is to determine whether their exists a schedule of given length $K$.

**Theorem 1.** *The* In-Pack *problem is NP-complete even when $r = e = 0$, i.e., when accounting only for memory to cache transfers.*

*Proof.* The In-Pack problem clearly belongs to NP. To show the completeness, we use a reduction from 3-Partition. Consider an arbitrary instance $\mathcal{I}_1$ of 3-Partition: given an integer $B$ and $3n$ integers $a_1, \ldots, a_{3n}$ (for all $i$, $\frac{B}{4} < a_i < \frac{B}{2}$), can we partition the $3n$ integers into $n$ triplets, each of sum $B$? We can assume that $\sum_{i=1}^{3n} a_i = nB$, otherwise $\mathcal{I}_1$ has no solution. The 3-Partition problem is NP-hard in the strong sense [3], which implies that we can encode all integers ($a_1$, ..., $a_{3n}$, $B$) in unary.

We build the following instance $\mathcal{I}_2$ of In-Pack:

- There are $q = n$ processors;
- There are $Bn$ tasks $(t_1, \ldots, t_{Bn})$;
- There are $Bn$ data inputs $X = (x_1, \ldots, x_{Bn})$;
- The dependence graph has $3n$ connected components (intuitively, one for each $a_i$). Each task in a connected component has two data inputs, as shown in Figure 4. Formally, let $A_i = \sum_{i=1}^{i-1} a_i$, then for $j \in \{1, \ldots, a_i\}$, the inputs of task $t_{A_i+j}$ are

$$I_{A_i+j} = \{x_{A_i+j}, x_{A_i+(j \mod a_i)+1}\}.$$

- Finally, we take $r = 0$ and $e = 0$ and ask whether there exists a schedule of execution time at most $K = wB$.

Clearly, $\mathcal{I}_2$ has a size polynomial in the size of $\mathcal{I}_1$ when $\mathcal{I}_1$ is written in unary. With $r = e = 0$, minimizing the execution time becomes the problem of minimizing:

$$\max_{1 \le j \le q} \Big| \bigcup_{i \in V_j} I_i \Big|. \tag{2}$$

In the following, we let $T_i = \{t_{A_i+1}, \ldots, t_{A_i+a_i}\}$ be the set of the $a_i$ tasks in the connected component corresponding to $a_i$. By construction,

$$\bigcup_{t_j \in T_i} I_j = \{x_{A_i+1}, \ldots, x_{A_i+a_i}\}. \tag{3}$$

Furthermore, let $i \neq i'$, then

$$\left( \bigcup_{t_j \in T_i} I_j \right) \bigcap \left( \bigcup_{t_j \in T_{i'}} I_j \right) = \emptyset. \tag{4}$$

We now prove that $\mathcal{I}_1$ has a solution if and only if $\mathcal{I}_2$ does. Assume first that $\mathcal{I}_1$ has a solution. For each triplet of the $n$ triplets $(a_i, a_j, a_k)$ of $\mathcal{I}_1$, we schedule $T_i$, $T_j$ and $T_k$ on a single processor. According to Equations (3) and (4), then the number of data inputs to copy on the cache on this processor is then equal

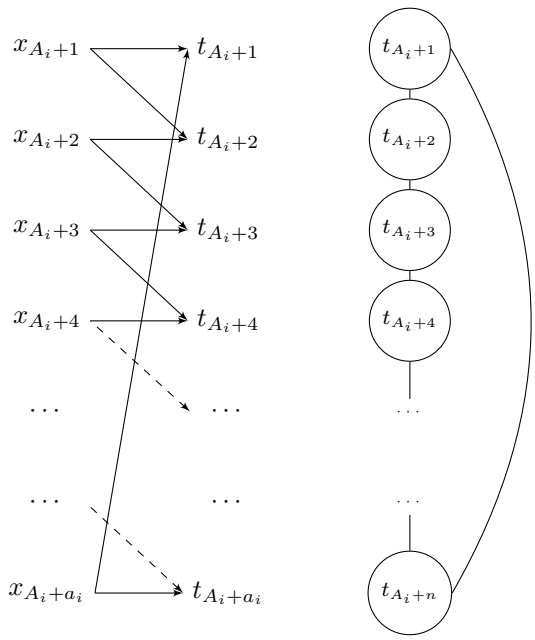


Figure 4: Input dependencies for the tasks (left) and its DAR graph (right). There are $3n$ such connected components.

to $a_i + a_j + a_k = B$. Hence this schedule has an execution time of $wB$. Hence a solution to $\mathcal{I}_2$.

Assume now that $\mathcal{I}_2$ has a solution. Since the execution time is $wB$, there are at most $B$ data input copied on each cache. Furthermore, because we have to copy all data input at least once, there are exactly $B$ data input copied on each cache. Finally, no data input is copied on two different caches, otherwise we would copy at least $nB+1$ data inputs, which would make at best an execution time of $w\frac{nB+1}{n}$.

Let us now show that for each set of tasks $T_i$, all the tasks of $T_i$ are scheduled on the same processor. By contradiction, assume that there exists $j < j'$ such that $t_{A_i+j}, t_{A_i+j'} \in T_i$, and $\pi(t_{A_i+j}) \neq \pi(t_{A_i+j'})$. Then we have two successive tasks between $t_{A_i+j}$ and $t_{A_i+j'}$ that are not mapped onto the same processor. Formally, there exists $j_{\min} > j$ such that

$$\pi(t_{A_i+j}) = \pi(t_{A_i+j_{\min}-1}) \neq \pi(t_{A_i+j_{\min}}) = \pi(t_{A_i+j'}).$$

Then $x_{A_i+j_{\min}-1}$ is an input common to both tasks $t_{A_i+j_{\min}-1}$ and $t_{A_i+j_{\min}}$, hence it has to be copied on both processors $\pi(t_{A_i+j})$ and $\pi(t_{A_i+j'})$, which contradicts the fact that each input is copied at most once.

This shows that each connected component of the task graph is mapped onto a single processor. Now each processor has execution time at most $wB$, hence it can be assigned at most three such connected components, because of the condition $\frac{B}{4} < a_i < \frac{B}{2}$ for all $i$. Finally, since the total number of writes is

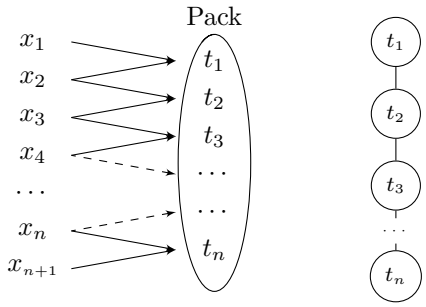


Figure 5: The dependency graph for a pack where $t_i$ only shares input with $t_{i-1}$ and $t_{i+1}$ (left), and its DAR graph (right).

$nB$, each processor has execution time exactly $wB$, and this gives a solution to $\mathcal{I}_1$. □

**A heuristic algorithm for the In-Pack problem.**

Fortunately, some task graphs are easier to schedule. Consider the task graph represented in Figure 5, where we have a long suite of tasks in a pack each with two inputs. We discuss in the next subsection when this structure arises in sparse triangular solution. In the In-Pack task graph on the right of the figure, there is an edge between $t_i$ and $t_{i+1}$ if they reuse data to solve for the unknowns in these tasks. This reuse is due to temporal locality of $x$ during concurrent execution of the tasks. Note that this graph corresponds to one connected component in Figure 4 but without the backward edge (and $n$ tasks instead of $a_i$). We will use this simple task graph in Section 4. Simply assume that $n$ is a multiple of $q$, $n = mq$. Then the static scheduling algorithm that assigns blocks of $m$ tasks to processors is optimal: it has an execution time $w \cdot (m+1) + e \cdot m + r \cdot (2m)$, and each of these terms is optimal. However, the model does not account for variability across processor speeds, a phenomenon to be expected in practice, and we transform the stator algorithm above into a dynamic heuristic as follows: To complete the tasks at the earliest, assign the available processors $c_1, c_2, \ldots, c_q$ to the tasks $t_1, t_2, \ldots, t_q$. At some point the tasks $t_1, t_2, \ldots, t_k$ would be completed. Assign the next task $t_{k+1}$ to processor $c_j$ if it is free. This assignment ensures sharing of $x$ in $L3$ between tasks $t_k$ and $t_{k+1}$.

Other scheduling techniques to enhance temporal locality could be envisioned, and we leave them for future work. In Section 4, we perform an experimental evaluation of the STS-k algorithm.

## 3.4 STS-k: A k-level Sub-structuring of Sparse $L \times x = b$ for Spatial and Temporal Locality of Data Accesses

We now propose a heuristic STS-k, that is informed by the algorithm to utilize temporal locality in the In-Pack assignment problem when its DAR is a *line* as shown in Figure 5. STS-k involves a further restructuring of super-rows in a pack so that its DAR will contain the line graph form.

Starting with the ordering across independent sets proposed earlier, namely a permutation $\pi$ in increasing order of pack size, consider a specific $P_k$ and its DAR($k$). Let $L^k x^k = b^k - L^{*j} x^{*j}$ correspond to the calculations across all tasks in $P_k$. The $x^k$ are calculated using $L^k$ and $b^k$ after previously computed components $x^{*j}$ and their corresponding elements in $L$, namely $L^{*j}$, are removed. Observe that DAR($k$) does not correspond to the graph of $A^k = L^k + {L^k}^T$. Instead it corresponds to the graph $G(\hat{A}^k)$ where $\hat{A}^k{}_{il} \neq 0$ if and only if $\hat{L}_{ij} \neq 0$ and $\hat{L}_{lj} \neq 0, j < i, l$ , i.e., two rows share a solution component $x_j$ from a previous pack corresponding to elements in $L^{*j}$. The matrix $\hat{A}^k$ need not be formed at all. However, observe that if $\hat{A}$ were to be tri-diagonal then $G(\hat{A}^k)$ would indeed be a line graph. Similarly, if $\hat{A}^k$ could be put in a sparse band-reduced form, then its corresponding $G(\hat{A}^k)$ would contain a line graph. We therefore reorder DAR($k$), and hence $G(\hat{A}^k)$ using Reverse Cuthill-McKee [1]. Following that, we permute $L_k$ according to this ordering. Other bandwidth reducing ordering schemes can also be used to reorder $\hat{A}^k$, we consider them in our future work.

The ordering across and within packs of the super-rows and their constituent rows, gives a 3-level sub-structure that we denote as $L^{STS-k} * x^{STS-k} = b^{STS-k}$. For a small sparse matrix from a fluid dynamics application $L$ is shown with coloring (9 colors), and restructured $L$ with STS-3 (4 colors) in Figure 6. Consider the last pack in each case; observe how the off-diagonal parts of our STS-k formulation are structured whereas these areas are disordered for the traditional scheme. These substructures in STS-k reflect the line graph form that is present in the reuse of solution components from previous packs.

The 3-level structure in STS-k maps very simply to the traditional compressed sparse row representation of sparse triangular solution formulation as shown in Algorithm 1 by using two additional sets of indices. An array $\texttt{index}_3$ of size $n_3$, the number of packs, points to the first super-row of each pack. Similarly, $\texttt{index}_2$ array indicates the first row of each super-row. These two arrays in addition to the traditional compressed sparse row format arrays, namely $\texttt{index}_1$, $\texttt{subscript}_1$, and $\texttt{value}_L$ complete the 3-level representation. Observe that the storage format is similar to what was developed earlier for parallel sparse matrix vector multiplication [4]. However, it differs in how it was specifically derived to expose parallelism and locality of data accesses for sparse triangular solution. These attributes derive in part from the original sparsity structure of $L$ and additionally from the effect of orderings across and within packs for spatial and temporal locality.

Thus far we have developed STS-k for $k$=3. However, depending on how many distinct levels there may be in a given NUMA multicore, additional groupings of super-rows into tasks could be considered for exposing locality in ways that could be potentially utilized for higher levels of sharing in the memory subsystem.

**Algorithm 1** STS-k with k=3 for sparse $L \times x = b$; the packs form the highest level of agglomeration at $k = 3$, followed by super-rows $k = 2$, and rows in the traditional compressed sparse row format $k = 1$, for a natural OpenMP implementation.

```
INPUT: index_3, index_2
INPUT: index_1, subscript_1, value_L, x, b
for i_3 = 1 to n_3 do
  #pragma omp parallel for schedule(runtime, chunk)
  for i_2 = index_3[i_3] to index_3[i_3 + 1] − 1 do
    for i_1 = index_2[i_2] to index_2[i_2 + 1] − 1 do
      temp_val = 0
      for j = index_1[i_1] to index_1[i_1 + 1] − 2 do
        temp_val+ = value_L[j] ∗ x[subscript_1[j]]
      end for
      x[i_1] = (b[i_1] − temp_val)/val_L[index_1[i_1 + 1] − 1]
    end for
  end for
end for
```

# 4 Experimental Results

In this section, we empirically evaluate the performance of STS-3 including comparisons with reference implementations with either level-set or coloring orderings. After a brief description of our evaluation approach, we consider the following. We first seek to characterize the levels of parallelism that are exposed when coloring or level-set orderings are applied to either the original matrix or its CSR-2 representation in STS-k. Next, we consider the incremental impact on performance solely from the $k$-level sub-structuring in STS-k for either ordering schemes. Finally, we consider how effective STS-k is in speeding-up the largest pack by increasing the locality of data accesses for its tasks.

## 4.1 Test Environment and Approach

*Matrix Test Suite.* Our test suite consists of the lower triangular parts of 12 sparse symmetric matrices selected from the University of Florida Sparse Matrix Collection [2]. These are listed in Table 1 with the number of rows $n$ ranging from 0.9 to 50.9 million, non-zeros ($nnz$) in the range 31.0 to 225.4 million and row densities ($nnz/n$) in the range 3.1 to 44.63.

*Multicore Architectures.* We perform tests on an Intel and on an AMD multicore node. The Intel node contains 4 Intel Xeon E7-8837 chips each containing 8 cores connected via QPI (Westmere-EX microarchitecture). The memory hierarchy of the system consists of L1, L2, L3 cache and 512 GB of DRAM. Each core has private 64 KB L1 and 256 KB L2 caches with access latencies of 4 and 10 cycles respectively [5]. The 24 MB L3 cache is shared among all 8 cores and it shows NUMA effects with latencies that can vary from 38 to 170 cycles depending on location. Furthermore, the memory access latencies are 175-290 cycles [5]. We

| **Matrix** | **n** | **nnz** | **nnz/n** |
|---|---|---|---|
| G1: ldoor | 952,203 | 42,493,817 | 44.63 |
| D1: rgg_n_2_21_s0 | 2,097,152 | 31,073,142 | 14.82 |
| S1: nlpkkt160 | 8,345,600 | 225,422,112 | 27.01 |
| D2: delaunay_n23 | 8,388,608 | 58,720,176 | 7.00 |
| D3: road_central | 14,081,816 | 47,948,642 | 3.41 |
| D4: hugetrace-00020 | 16,002,413 | 64,000,039 | 4.00 |
| D5: delaunay_n24 | 16,777,216 | 117,440,418 | 7.00 |
| D6: hugebubbles-00000 | 18,318,143 | 73,258,305 | 4.00 |
| D7: hugebubbles-00010 | 19,458,087 | 77,817,615 | 4.00 |
| D8: hugebubbles-00020 | 21,198,119 | 84,778,477 | 4.00 |
| D9: road_usa | 23,947,347 | 81,655,971 | 3.41 |
| D10: europe_osm | 50,912,018 | 159,021,338 | 3.12 |

Table 1: Test suite of matrices with dimension $n$, the number of non-zeroes $nnz$, and the average row densities $nnz/n$.

denote this node by Intel in the rest of the paper.

The AMD node consists of 2 twelve-core AMD 'MagnyCours' processors [7]. The memory subsystem of this processor has L1, L2, L3 cache and 64 GB of DRAM. Each core has private 64 KB L1 and 512 KB L2 cache and a 6 MB L3 cache is shared among 6 cores. This processor is denoted by AMD in the rest of the paper.

The value of $k$ in STS-k is determined as follows. One value of k is reserved for set of independent tasks, i.e., at the pack level. The value of k-1 depends on a particular architecture and the number of distinct levels of caches with well-differentiated latencies from NUMA effects. In our test systems, L1 and L2 are private with relatively low latencies while L3 is shared with a large variation in latencies, leading to k-1=2, with third value indicating pack level grouping. So we choose $k = 3$ and evaluate the performance of STS-3 on both Intel and AMD processors. In practice, the sensitivity of performance to the value of k could be tested by trying +/-1 values. Additionally, to correspond to bigger L2 cache on AMD, super-rows are consisted of 320 rows of $G_1$ while super-rows have 80 rows of $G_1$ on Intel.

*Sparse Triangular Solution Schemes.* We consider sparse triangular solution using the compressed sparse row format (CSR) with either level-set or coloring orderings as the reference schemes for comparisons. These are labeled as CSR-LS and CSR-COL in the remainder of this paper. Our STS-k formulation with $k = 3$ and coloring is denoted by STS-3 or equivalently CSR-3-COL. Its alternative formulation with level-set ordering is indicated as CSR-3-LS.

The reference implementations, namely CSR-LS and CSR-COL, tend to be far more sensitive to the initial ordering of the matrix than STS-k. They typically perform best when the matrix is presented in the Reverse Cuthill-McKee [1], i.e., RCM, ordering. We therefore use the RCM ordered matrix as the input to all schemes including STS-k. Furthermore, for all schemes, fol-

lowing the application of either level-set or coloring, the packs are arranged in increasing order of their sizes and the rows within packs are renumbered contiguously. Level-set orderings are constructed starting with a vertex of largest degree, because we have observed this leads to fewer and larger packs. Consequently, they provide near best case instances for comparisons with coloring in regard to effectiveness in extracting parallelism. Colorings were obtained using the Boost library [10]. These steps ensure that differences between the two pairs, CSR-3-LS and CSR-LS, and CSR-3-COL (or STS-3) and CSR-COL, are removed except when that are by design.

All methods were compiled using Intel's icc 15.0.0 (Intel) and 15.0.1 (AMD) with -O3 optimization. We use Intel's OpenMP 4.0 with thread affinity set via KMP_AFFINITY = compact, and with KMP_LIBRARY = throughput. Tests with CSR-COL and CSR-LS used scheduling with
OMP_SCHEDULE = $dynamic, 32$ because they resulted in the best performance. For similar reasons, tests with STS-3 (CSR-3-COL) and CSR-3-LS used OMP_SCHEDULE = $guided, 1$. We consider execution time in seconds measured as the average of 10 sparse triangular solution repeats. Overheads of pre-processing for all schemes are ignored. This is an accepted approach because such systems are solved for large sequences of right-hand side vectors within an application with the pre-processing costs getting amortized over all these repetitions.

## 4.2 Effectiveness Level-Set and Coloring Orderings in Extracting Parallelism

We would like to consider how effective level-set and coloring orderings are in regard to exposing parallelism. Figure 7 shows how the four methods cluster over all 12 matrices along two key dimensions, namely the number of packs and the number of solution components that are calculated on average per pack. The clustering indicates that coloring is significantly more effective than level-set at exposing parallelism; the number of packs using level-set are larger while their sizes are larger using coloring, both by several orders of magnitude (note log scale). Furthermore, applying CSR-3, as we propose for the STS-3 formulation, causes small decreases in the number of packs with slight increases in their sizes. This is evident from the positioning of the cluster centroids for CSR-3-COL (STS-3) and CSR-3-LS slightly below and to the left of those for CSR-COL and CSR-LS respectively.

The total work in sparse triangular solution is proportional to the number of nonzeroes, where work is a fused multiple-add instruction. The percentage of this total work contained in a few of the largest packs is another measure of the degree of parallelism. This measure is particularly important because it reflects how latencies could be masked by parallelism subject to bandwidth constraints (Little's Law). Additionally, when the total work is scattered across many packs, synchronization costs between packs can be substantial, degrading performance. The percentage of parallel work in the 5 largest packs is shown in Figure 8 further substantiating coloring to be the ordering of choice.

The measures of parallelism that we have reported thus far in Figures 7 and 8, are independent of the multicore architecture. Although they predict a sizable gap in performance between schemes that use level-set versus coloring, hardware and OpenMP attributes will likely modify the actual gaps that will be presented in terms of execution times. We consider this aspect next.

Let $T(mat, \text{method}, q)$ indicate the execution time for a matrix *mat*, with $q$ cores, and *method* as one of STS-3, CSR-3-LS, CSR-LS and CSR-COL. To compare across both coloring and level-set orderings, we use CSR-LS as the reference method relative to which we report performance measures. We define *parallel speedup* for a specific method and matrix *mat* on $q$ cores as:

$$\frac{T(mat, \text{CSR-LS}, 1)}{T(mat, \text{method}, q)}.$$

The parallel speedup factors are shown in Figure 9 for 16 cores on Intel and 12 cores on AMD processors; geometric means are shown as horizontal lines. Key observations include the following:

- STS-k or equivalently CSR-3-COL is readily the best with a factor 6 (Intel) and 4 (AMD) improvement relative to the reference CSR-LS implementation. CSR-COL which also uses coloring is next in rank for Intel and CSR-3-LS is next in rank for AMD. However, there is a big gap in performance of CSR-3-COL and CSR-COL which is not apparent from the data in Figures 7 and 8. The fact that STS-k outperforms CSR-COL significantly is largely a consequence of the $k$-level subs-structuring and its effect in decreasing data access latencies.

- Interestingly, despite the very large gap in the levels of parallelism between CSR-COL and CSR-3-LS as shown in Figures 7 and 8, their parallel speedup values are very close, with mean values of approximately 2.85 (Intel) and 1.8 (AMD) relative to the reference CSR-LS implementation. This is largely a result of the effect of the $k$-level sub-structuring that we have proposed which reduces the latencies of data access times.

## 4.3 Performance: Evaluating Incremental Impacts from k-Level Sub-Structuring

We now seek to evaluate the incremental impacts on parallel performance solely from the $k$-level sub-structuring in STS-k. We focus on the performance of CSR-3-COL relative to CSR-COL and CSR-3-LS relative to CSR-LS. We define *relative speedup* for coloring orderings as:

$$\texttt{relative speedup}(\text{coloring}) = \frac{T(mat, \text{CSR-COL}, q)}{T(mat, \text{STS-3}, q)},$$

note that STS-3 is equivalent to CSR-3-COL.

The *relative speedup* for level-set orderings is defined as:

$$\texttt{relative speedup}(\text{level-set}) = \frac{T(mat, \text{CSR-LS}, q)}{T(mat, \text{CSR-3-LS}, q)}.$$

A value greater than 1 of *relative speedup* indicates the factor by which the $k$-level scheme outperforms the reference method. Figure 10 indicates that our $k$-level sub-structuring is indeed beneficial on 16 cores of Intel with a factor of 2.2 and on 12 cores of AMD with a factor of 2.19 improvement on average over all matrices when applied with coloring. Similarly, Figure 11 shows that our $k$-level sub-structuring is beneficial with level-set ordering, with a factor of 1.4 and 1.5 improvement on average on Intel and AMD respectively.

We next consider the *relative speedup* measured using the total time over all the matrices in the test suite. We denote the sum of execution time over all matrices for a method on q core as $T(*, \text{method}, q)$, where $q$ ranging from 1-32 on Intel and 1-24 on AMD. Using total execution time, *relative speedup* for coloring is defined as:

$$\frac{T(*, \text{CSR-COL}, q)}{T(*, \text{STS-3}, q)},$$

and *relative speedup* for level-set ordering is defined as:

$$\frac{T(*, \text{CSR-LS}, q)}{T(*, \text{CSR-3-LS}, q)}.$$

Figures 12 and 13 show the factors by which our $k$-level structuring scheme outperforms the reference methods on all the core counts and over the whole test suite. We observe that $k$-level structuring outperforms by a factor of 2 with coloring and a factor of 1.4 for level-set on $8 - 32$ cores of Intel. The $k$-level structuring also outperforms by a factor of 2.2 with coloring and 1.5 with level-set on $6 - 24$ cores of AMD. These results show that our $k$-level sub-structuring is indeed very effective and can reduce execution times by one-third to one-half.

## 4.4 Effects of Enhanced Levels of Locality on Execution Times of Largest Pack with Coloring

Our findings thus far confirm the superiority of coloring over level-set orderings and demonstrate substantial performance gains from the $k$-level sub-structuring. In particular, we would like to assess to what extent the latter contribute towards enhanced levels of spatial and temporal locality in processing a pack. To measure this we will need to remove other factors such as synchronization costs between packs that can be significant as observed by Wolf et. al [12]. Additionally, we need to account for the fact that there is not a one-to-one correspondence between packs for a given matrix with CSR-COL and CSR-3-COL.

Let $t(\text{method}, q)$ denote the execution time for the largest pack $P*$ for a given method on $q$ cores scaled by its number of unknowns (solution components). We define the *parallel speedup per unknown* as the ratio:

$$\frac{t(\text{CSR-COL}, q)}{t(\text{STS-3}, q)}.$$

The *parallel speedup per unknown* on 16 cores of Intel and 12 cores of AMD is shown in Figure 14. Observe that on average, this measure is high at 1.75 on Intel and 2.12 on AMD across all the matrices in the test suite. Thus the performance gain of STS-k is due to our STS-k formulation.

# 5 Conclusions

We have developed a multilevel sub-structured formulation of sparse triangular solution to increase locality of data accesses and thus reduce effective access latencies in NUMA multicores. Our data affinity and reuse graph model of the underlying computations informs how packs of independent tasks may be scheduled for enhanced levels of reuse from temporal locality of accesses. The observed performance gains are both promising and significant.

We plan to consider more complex NUMA systems where STS-k with values of $k$ greater than 3 may be appropriate. Such formulations could potentially be obtained through further groupings of super-rows based on dependencies before coloring orderings are applied or after they are applied to define tasks in terms of collections of super-rows.

We expect to study the problem of NUMA-aware schedules when data affinity and reuse graphs are constructed to span more than one pack. While optimal versions will likely be NP-complete, heuristic approaches and special cases of graphs can inform alternate sub-structuring schemes that could reduce synchronization overheads and increase locality.

Current trends in microprocessor architecture indicate increasing core counts with complex NUMA hierarchies. These NUMA effects are expected to become more pronounced when network-on-chip (NoC) architectures are considered. In this context, we see potential for further performance benefits through schemes for parallel sparse computations such as the ones we have developed, with multiple levels of sub-structuring for enhancing locality of accesses and the explicit consideration of data reuse for the scheduling of tasks.

# References

[1] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, ACM '69, pages 157–172, New York, NY, USA, 1969. ACM.

[2] T. A. Davis. University of Florida Sparse Matrix Collection. *NA Digest*, 92, 1994.

[3] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co, 1979.

[4] H. Kabir, J. Booth, and P. Raghavan. A multilevel compressed sparse row format for efficient sparse computations on multicore processors. In *High*

*Performance Computing (HiPC), 2014 21st International Conference on*, pages 1–10, Dec 2014.

[5] D. Molka, D. Hackenberg, R. Schone, and M. S. Muller. Memory Performance and Cache Coherency Effects on an Intel Nehalem Multiprocessor System. In *Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques*, PACT '09, pages 261–270, Washington, DC, USA, 2009. IEEE Computer Society.

[6] M. Naumov. Incomplete-LU and Cholesky Preconditioned Iterative Methods Using cuSPARSE and cuBLAS. Technical report, Nvidia, 2011.

[7] NERSC. Hopper at NERSC, https://www.nersc.gov/users/computational-systems/hopper.

[8] J. H. Saltz. Aggregation methods for solving sparse triangular systems on multiprocessors. *SIAM J. Sci. Stat. Comput.*, 11(1):123–144, Jan. 1990.

[9] R. Schreiber and W. P. Tang. Vectorizing the conjugate gradient method. *Symposium on CYBER 205 Applications*, 1982.

[10] J. G. Siek, L.-Q. Lee, and A. Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual (C++ In-Depth Series)*. Addison-Wesley Professional, 2001.

[11] B. Suchoski, C. Severn, M. Shantharam, and P. Raghavan. Adapting Sparse Triangular Solution to GPUs. In *Proceedings of the 2012 41st International Conference on Parallel Processing Workshops*, ICPPW '12, pages 140–148, Washington, DC, USA, 2012. IEEE Computer Society.

[12] M. M. Wolf, M. A. Heroux, and E. G. Boman. Factors impacting performance of multithreaded sparse triangular solve. In *Proceedings of the 9th International Conference on High Performance Computing for Computational Science*, VECPAR'10, pages 32–44, Berlin, Heidelberg, 2011. Springer-Verlag.

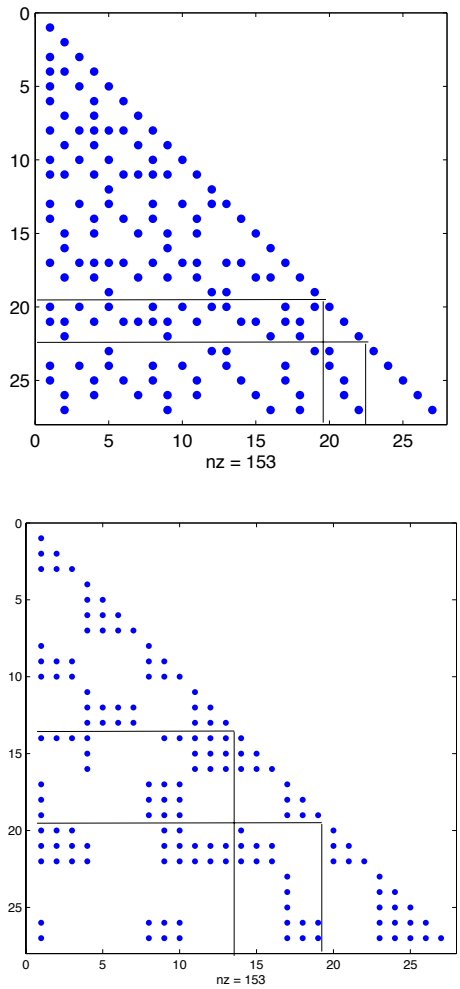


Figure 6: L is shown with coloring (top) and STS-k (bottom); the two biggest packs are are enclosed in boxes along the diagonal. Observe the sub-structures that are revealed in STS-k that correspond to reuse of solution components among tasks in each pack.

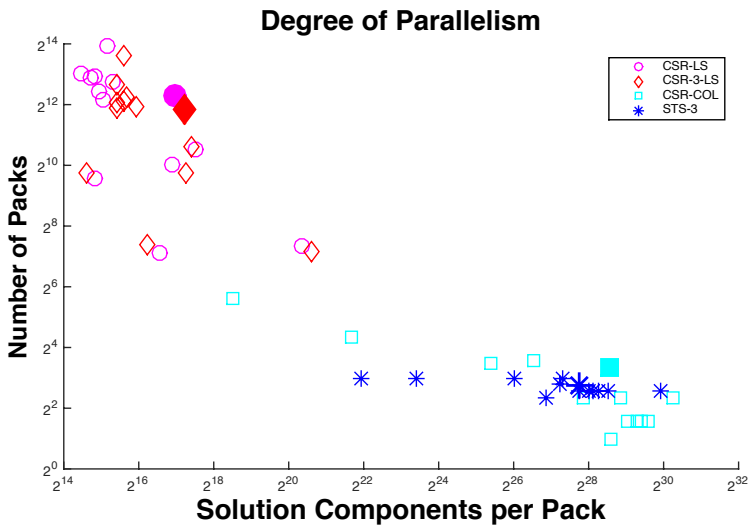


Figure 7: Each point represents observed values of the two measures of parallelism, the number of packs and the umber of solution components per pack, for one of four methods on a matrix. Observe that across all matrices in the test suite, clusters for level-set (with the pair CSR-LS and CSR-3-LS) and coloring with the pair (CSR-COL and CSR-3-COL) are very well separated. Further, their centroids clearly show coloring to be vastly superior (note the log scale).

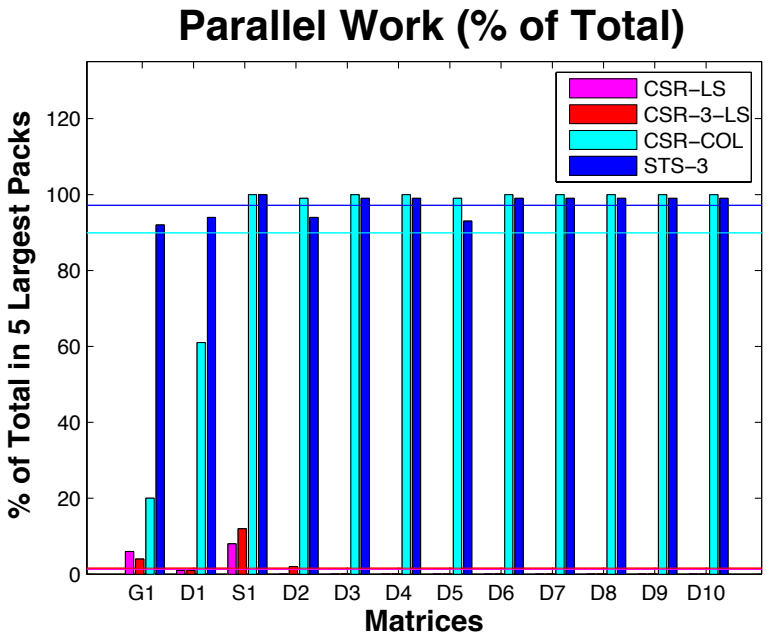


Figure 8: The percentage of total work in the 5 largest packs with mean value shown as lines. Observe that CSR-COL and STS-3 (CSR-3-COL) in the largest 5 packs contain over 90% of total work, where as in CSR-3-LS and CSR-LS the largest 5 packs contain less than 5% of total work.

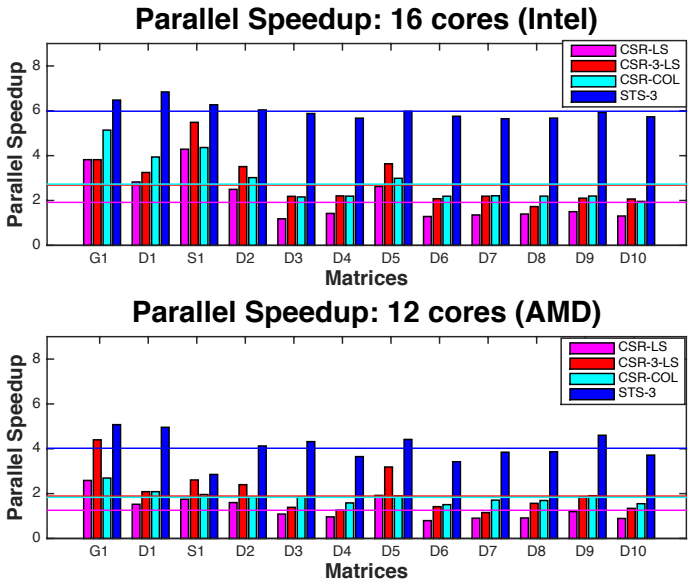


Figure 9: Parallel speedup of STS-3, CSR-COL and CSR-3-LS compared to CSR-LS on 16 cores Intel (top) and 12 cores AMD (bottom). The horizontal lines show the mean speed up for the entire test suite.

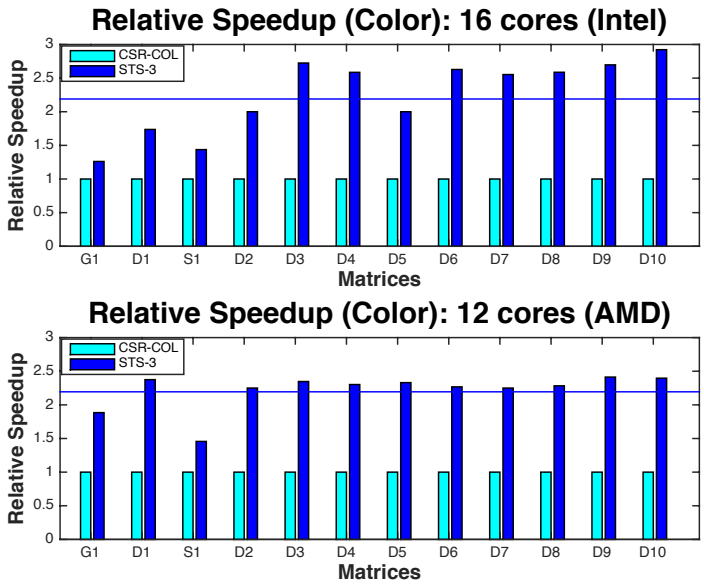


Figure 10: Relative speedup of STS-3 compared to coloring on Intel (top) and AMD (bottom). The horizontal line shows the mean speedup of STS-3.

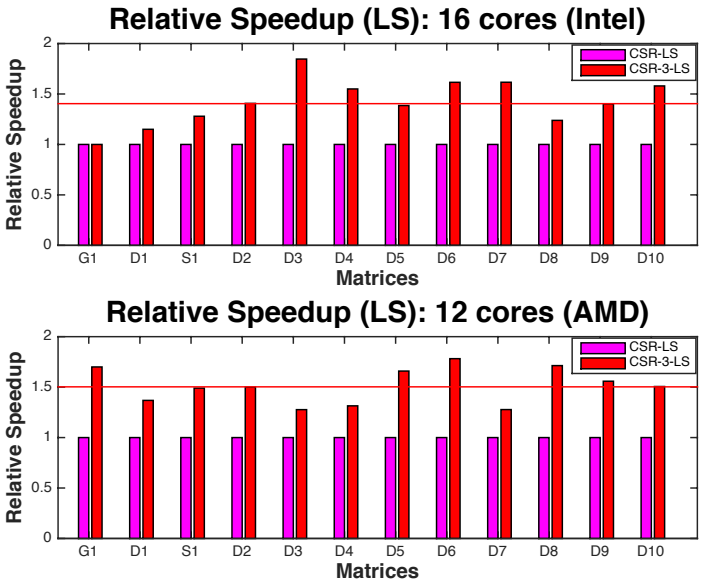


Figure 11: Relative speedup of CSR-3-LS compared to CSR-LS on Intel (top) and on AMD (bottom). The horizontal line shows the mean speedup of CSR-3-LS.

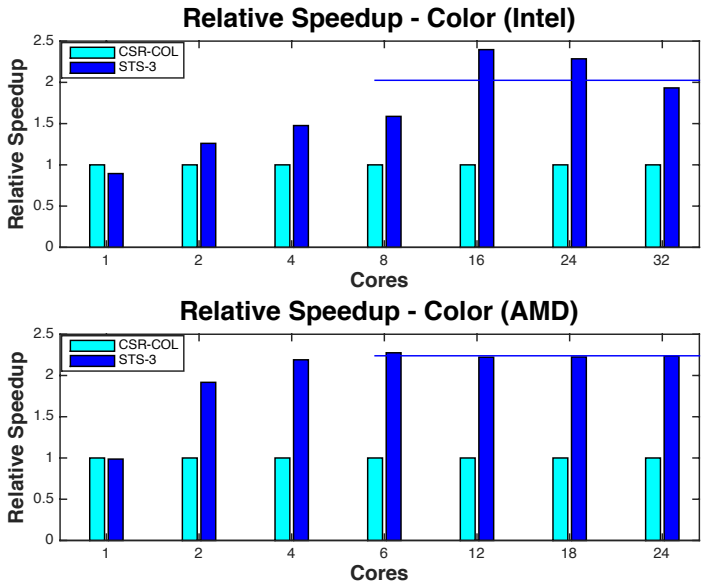


Figure 12: Relative speedup of STS-3 compared to CSR-COL for the whole test suite on 1-32 cores of Intel (top) and 1-24 cores of AMD (bottom). The horizontal line shows mean value for 8-32 cores (Intel) and 6-24 cores (AMD).

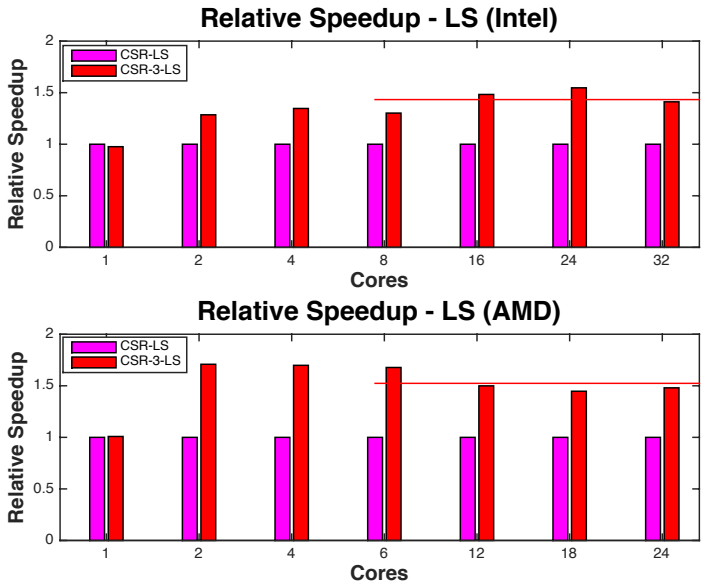


Figure 13: Relative speedup of CSR-3-LS compared to CSR-LS for the whole test suite on 1-32 cores of Intel (top) and 1-24 cores of AMD (bottom). The horizontal line shows the mean value for 8-32 cores (Intel) and 6-24 cores (AMD).

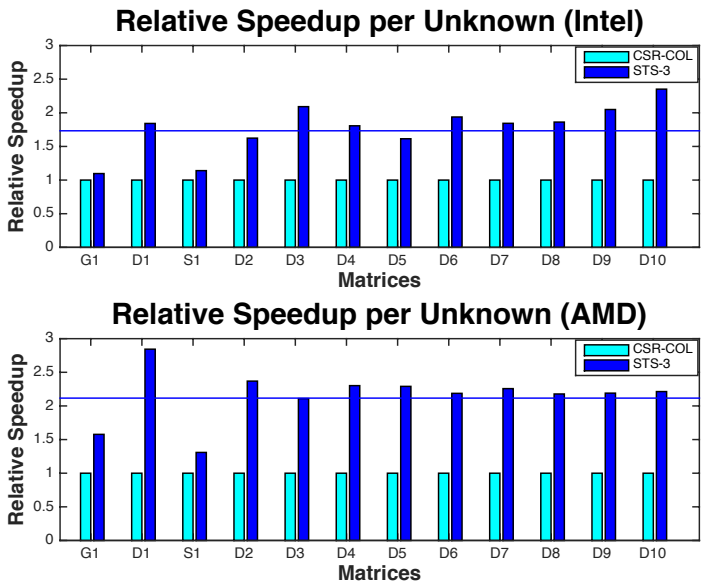


Figure 14: Relative speedup of STS-3 compared to CSR-COL for processing one solution component on Intel (top) and on AMD (bottom). The horizontal line shows the mean value of STS-3.

**RESEARCH CENTRE**
**GRENOBLE – RHÔNE-ALPES**

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399