Eliminating Memory Fragmentation within Partitionable SIMD/SPMD Machines

Mark A. Nichols, Member, IEEE, Howard Jay Siegel, Fellow, IEEE, Henry G. Dietz, Russell W. Quong, Member, IEEE, and Wayne G. Nation, Member, IEEE

Abstract—Efficient data layout is an important aspect of the compilation process. A model for the creation of “perfect” memory maps for large-scale parallel machines capable of user-controlled partitionable SIMD/SPMD operation is developed. The term “perfect” implies that no memory fragmentation occurs and ensures that the memory map size is kept to a minimum. The major constraint on solving this problem is one based on the single program nature of both the SIMD and SPMD modes of parallelism. Specifically, it is assumed all processors within the same submachine use identical addresses to access corresponding data items in each of their local memories. Necessary and sufficient conditions are derived for being able to create “perfect” memory maps and these results are applied to several partitionable interconnection networks.

Index Terms—Compilers, interconnection networks, memory fragmentation, parallel processing, partitionable machines, PASM, SIMD, SPMD.

I. INTRODUCTION

Many parallel machines with physically distributed memory (i.e., each processor is physically paired with a memory module) have been built, such as the SIMD Connection Machine [17], [38] and the MIMD NCube [16]. When designing large-scale parallel machines (e.g., 25 to 216 processors), it is worthwhile to consider giving the system the ability to dynamically group together their processors to form independent or cooperating submachines of various sizes [35]. Machines with this capability are referred to as partitionable. The class of systems considered here is large-scale, partitionable, physically distributed memory, parallel machines. The research presented is based on user-controlled partitioning of cooperating submachines that is specified at compile-time and can occur at instruction level granularity.

Two modes of parallelism that are focused on in this paper are SIMD and SPMD. In the SIMD (single instruction—multiple data) mode [14] of parallelism, a single control unit broadcasts instructions and each active processor operates on its own data. In the SPMD (single program—multiple data) [9], [10] restriction of the MIMD (multiple instruction—multiple data) mode [14] of parallelism, each processor functions independently, however all processors execute the same program. When computing in SPMD mode, the processors will more than likely be operating on different data sets, so they may be taking different control paths through the program. These two modes can be combined into a single hybrid system; e.g., OPSILA is a SIMD/SPMD distributed memory system that has been built [2], [3], [11]. The advantages of combining both modes into a single system are discussed in [5].

Each submachine in a system in the class being considered is capable of SIMD and/or SPMD operation. Thus, these results apply to many types of parallel architectures: multiple-SIMD machines [25] (i.e., a partitionable machine with multiple control units where each submachine can operate as an independent SIMD system), partitionable SPMD machines (i.e., a partitionable machine where each submachine operates in the SPMD mode of parallelism), MIMD machines when operating in partitionable SPMD mode, partitionable SIMD/SPMD machines (i.e., a partitionable machine where each submachine is independently capable of dynamically switching between the SIMD and SPMD modes at instruction level granularity), and partitionable SIMD/MIMD machines when operating in partitionable SIMD/SPMD mode.

Some parallel machines that have been built with physically distributed memory (i.e., each processor has an associated local memory) and some form of partitioning are: the BBN Butterfly [8], IBM RP3 [28], Connection Machine [17], [38], iPSC Hypercube [24], NCube system [16], and PASM [6], [13], [34], [35]. Of these, the Butterfly, iPSC Hypercube, NCube system, and RP3 are MIMD machines capable of partitionable SPMD mode. The Connection Machine hardware can support partitionable SIMD mode because it is a multiple-SIMD machine. Also note that the proposed MAP [25] machine and the original design for the Illiac IV [4] were multiple-SIMD architectures. (A formal model of multiple-SIMD machines is presented in [29].) PASM is a partitionable SIMD/MIMD machine (capable of partitionable SIMD/SPMD operation). This research is motivated by compiler development for the PASM prototype [22], [23].

1These machines use a shared memory addressing scheme, but the memories are physically distributed with each memory associated with one local processor.
The parallel applications addressed by this paper consist of an overall task that can be decomposed into many subtasks, where each subtask can employ parallelism and is mapped onto a specific submachine by the user. Typically, some subtasks will be able to execute concurrently, while others must adhere to temporal precedence constraints. The user specifies, at compile-time, the partitioning of the system into submachines, the mapping of subtasks onto submachines, and the temporal ordering of the subtasks. While automatic system reconfiguration of partitionable machines for improved performance and fault tolerance is a long term goal [7], much research needs to be done. For the class of systems under consideration, explicit user control will be appropriate and necessary for the near term future to utilize these systems more effectively and to help gain the prerequisite knowledge for automatic reconfiguration.

Efficient data layout is an important aspect of the compilation process for large-scale partitionable machines. A memory map specifies how the data/code for different subtasks is distributed among the memory modules associated with the processors. A model for the creation of fragment-free (perfect) memory maps is developed here. Given a parallel task to be performed, it is desirable to have the whole task reside in primary memory for its entire execution, avoiding delays resulting from paging. The primary memory utilization problem with respect to the set of processors is dealt with based on the single program nature of both the SIMD and SPMD modes of parallelism. Specifically, it is assumed that due to the single instruction/program operation of these modes, when a particular variable address is needed in a program, the same absolute address is used by all processors in a submachine to access the corresponding data item allocated in each of their local memories (a variable’s value is permitted to differ among the processors in a submachine). This is also assumed for branch target addresses in SPMD mode. These assumptions imply that if the code/data segments are packed into the memory maps in an arbitrary order, fragmentation within the memory maps can occur. Worst case, this fragmentation grows exponentially with the number of processors because it is directly related to the number of distinct submachines that can be utilized within a system.

This paper provides necessary and sufficient conditions for a partitionable SIMD and/or SPMD machine to have fragment-free (perfect) memory maps for the execution of any parallel tasks. Because machine partitionability is usually a function of a machine’s interconnection network, a study is also performed on a few networks to determine whether machines that employ these networks satisfy these necessary and sufficient conditions. The generation of processor memory maps is the primary focus; therefore, the treatment of SPMD mode code and data segments and SIMD mode data segments is considered. It is assumed that all such code and data for a given task will be stored in the processor memories. The code segments for SIMD mode are not kept within processor memories so, in general, they will not be discussed. Some tasks result in the memory maps for a partitionable SIMD/SPMD task is important. If the entire memory requirement for such a task can be satisfied by primary memory, then there is no need to go to a secondary memory source. Also, hardware cost is a factor, especially within large-scale parallel processing systems. If a scheme always generates processor memory maps that contain no fragmentation, then the amount of memory required for an “average-sized” task is minimized. This provides system architects the option of implementing smaller local memories or allowing users to operate on larger data sets.

Section II develops a model for machine partitioning, partitionable SIMD/SPMD addressing constraints, and the creation of “perfect” memory maps. Necessary and sufficient conditions on a partitionable machine for the creation of “perfect” memory maps are derived in Section III. Section IV applies these results to machines that employ different types of partitionable interconnection networks.

II. MEMORY MODEL FOR PARTITIONABLE SIMD/SPMD MACHINES

The formalism presented within this section and the next requires some concepts from discrete mathematics [1], [15], [36] that are reviewed first. Given set A and binary relation \( \leq \) on A, if \( \leq \) is reflexive (i.e., if \( x \leq A, x \leq x \)), antisymmetric (i.e., if \( x, y \in A, x \leq y \land y \leq x \implies x = y \)), and transitive (i.e., if \( x, y, z \in A, x \leq y \land y \leq z \implies x \leq z \)) then the ordered pair \((A, \leq)\) defines a partial ordering on A. The Hasse diagram of the partial ordering \((A, \leq)\) is the directed acyclic graph (DAG) constructed by placing an arc from node \( x \) to node \( y \) if \( x \leq y, x \neq y \), and there does not exist a \( z \) such that \( x \leq z \leq y \), for \( x, y, z \in A \). The ordered pair \((A, \leq)\) defines a total ordering on A if \((A, \leq)\) is a partial ordering where for all \( a, b \in A \) either \( a \leq b \) or \( b \leq a \). A topological sort of partial ordering \((A, \leq)\) is a total ordering \((A, \leq)\) that has embedded within it the partial ordering; i.e., if \( a, b \in A \), then \( a \leq b \implies a \leq b \), while \( a \leq b \implies a \leq b \). A tree is a DAG with exactly one arc entering a node (except for the root node).

The remainder of this section defines a model used to describe machine partitioning capabilities, partitionable SIMD/SPMD addressing constraints, and the creation of “perfect” memory maps. The first set of definitions covers partitionable machines capable of operating in SIMD and/or SPMD mode.

1) A processing element (PE) is a processor/memory pair.

2) A processing element group (PEG) is a set of one or more PE’s (this paper assumes that the set of PE’s within a given PEG is fixed for the duration of a parallel task [29]).

3) A partitionable machine M includes a set of PEG’s, i.e., \( M = \{PEG_0, PEG_1, \ldots, PEG_{|M|-1}\} \). The term \(|M|\) denotes the cardinality of set \( M \).

As the start of a running example that continues throughout this section, consider a machine \( M \) that has four PEG’s, i.e., \( M = \{PEG_0, PEG_1, PEG_2, PEG_3\} \).

4) A submachine \( S \) of partitionable machine \( M \) is a subset of \( M \), i.e., \( S \subseteq M \), that is executing a given subtask.

Some of the submachines that can be constructed from machine \( M \) are \( S_0 = M, S_1 = \{PEG_0, PEG_1\}, S_2 = \{PEG_1, PEG_2\}, S_3 = \{PEG_2, PEG_3\}, S_4 = \{PEG_0\}, S_5 = \{PEG_1\}, S_6 = \{PEG_2\}, S_7 = \{PEG_3\}. \)
5) In a partitionable SIMD machine (also called multiple-SIMD machine), there are multiple control units and the PE’s of each submachine are broadcast all of the code they execute by an independent physical or virtual control unit. All PE’s within a submachine are implicitly synchronized after every instruction.

6) In a partitionable SPMD machine, each PE functions independently using its own program counter and all PE’s in a given submachine are executing the same program. There is no implicit synchronization as in the partitionable SIMD machine case.

7) In a partitionable SIMD/SPMD machine, all submachines can independently and dynamically switch between SIMD mode and SPMD mode at instruction level granularity.

The following collection of definitions focuses on machine partitioning, tasks, and the set of all possible submachines that can be generated on a partitionable machine. The submachine set is instrumental in the determination of whether a partitionable machine supports fragment-free memory maps.

8) A partitioning $P$ of machine $M$ is a set of submachines, $P = \{S_0, S_1, \ldots, S_{|P|-1}\}$, such that each PEG of $M$ is in exactly one $S_i$, for $0 \leq i < |P|$, i.e., $P$ consists of mutually disjoint subsets that cover $M$.

A collection of four partitionings for the example machine $M$ defined above is

\[
\begin{align*}
P_0 &= \{S_0\} = \{M\} \\
P_1 &= \{S_1, S_3\} = \{\text{peg}_{00}, \text{peg}_{11}\} \\
P_2 &= \{S_4, S_2, S_7\} = \{\text{peg}_{01}, \text{peg}_{12}, \text{peg}_{23}\} \\
P_3 &= \{S_6, S_0, S_7\} = \{\text{peg}_{02}, \text{peg}_{13}, \text{peg}_{23}\}.
\end{align*}
\]

9) The partitioning set $R$ for machine $M$ is the set of all possible partitionings permitted within $M$, i.e., $R = \{P_0, P_1, P_2, P_3\}$. This may be a function of system hardware and/or software constraints, e.g., the ways in which the interconnection network may be partitioned [31], [32].

For the example machine $M$, define the partitioning set to be: $R = \{P_0, P_1, P_2, P_3\}$, where $P_0$, $P_1$, $P_2$, and $P_3$ are specified above.

10) A submachine set $Q$, for machine $M$ with partitioning set $R$ is the set of all possible submachines permitted within $M$, i.e., $Q = \{S_0, S_1, \ldots, S_{|Q|-1}\} = \bigcup_{i=0}^{R-1} P_i$. Thus, for $0 \leq i < |R|, P_i \subseteq Q$.

The submachine set corresponding to partitioning $R$ for the running example is $Q = \bigcup_{i=0}^{3} P_i = \{S_0, S_1, \ldots, S_7\}$, where $P_0, P_1, P_2$, and $P_3$ and $S_0, \ldots, S_7$ are as defined above.

11) A submachine set $Q$ is nested if for all $S_x, S_y \in Q, (S_x \cap S_y) \neq \emptyset$ implies that either $S_x \subseteq S_y$ or $S_y \subseteq S_x$.

Continuing the example, the Hasse diagram $\bar{R}$ of $(Q, \subseteq)$ is shown in Fig. 1. Because $(S_1 \cap S_2) = \{\text{peg}_{30}\}$ and the fact that $S_1$ and $S_2$ are not subsets of one another, the submachine set $Q$ is not nested.

If a different partitioning $R' = \{P_0, P_1, P_3\}$ is used (i.e., $R' = R - \{P_2\}$), the new submachine set is $Q' = P_0 \cup P_1 \cup P_3 = \{S_0, S_1, S_2, S_3, S_5, S_6, S_7\}$ (i.e., $Q' = Q - \{S_4\}$). The Hasse diagram $\bar{R}'$ of $(Q', \subseteq)$ is shown in Fig. 2. $Q'$ is nested.

12) A task $E$ performed on machine $M$ is a temporal sequence of partitionings specified by $E = (\pi_0, \pi_1, \ldots, \pi_{|E|-1})$

where each $\pi_i$ is a partitioning of $M$ (i.e., $\pi_i \in R$). $M$ begins execution of $E$ by being configured for partitioning $\pi_0$. For the duration of $E$, $M$ successively reconfigures itself at execution-time for each of the $|E|$ partitionings in $E$.

For the example machine $M$ with partitioning set $R$, one possible task is $E = (P_0, P_1, P_0, P_1)$ (i.e., $\pi_0 = P_0, \pi_1 = P_1, \pi_2 = P_0, \pi_3 = P_1$).

13) A task submachine set $V$ for task $E$ is the set of all submachines used by $E$, i.e., $V = \{S_0, S_1, \ldots, S_{|V|-1}\} = \bigcup_{i=0}^{|E|-1} \pi_i$. Note that $V \subseteq Q$.

For the task $E$ within the running example, $V = \bigcup_{i=0}^{3} \pi_i = P_0 \cup P_1 = \{S_0, S_1, S_3\}$.

Within the next set of definitions, memory maps, “perfect” memory maps, code/data segments, and addressing constraints are formally defined.

14) A submachine segment $D_i$ for submachine $S_i \in Q$ is a space of $d_i$ words $D_i[0 \ldots d_i-1]$ that contains the SIMD data segments and SPMD data and code segments for the different subtasks executed by $S_i$ within a particular parallel task. The notation “$a \ldots b$” refers to all addresses between $a$ and $b$ inclusive. The data values held in this space can differ among PE’s in $S_i$, however it is assumed that $d_i$ is the same for all PE’s in $S_i$. It is important to note that for a particular task $E$, a given $S_i$ may be contained in multiple $\pi_j$’s, $0 \leq j < |E|$. Whenever this exact submachine $S_i$ (with no more or no fewer PEG’s) is used within any partitionings ($\pi_j$’s) within a given task $E$, all SIMD data and SPMD data and code segments that are used by the submachine $S_i$ are included within its submachine segment $D_i$.

Continuing the running example, given all submachines contained in $Q'$ and defined above for machine $M$, Fig. 3 provides a table of the submachines’ segments ($D_i$’s) and hypothetical corresponding sizes ($d_i$’s).

15) For $\text{peg}_{i} \in M$, a PEG memory map $U_i$ is a fixed size array of $u_i$ locations $U_i[0 \ldots u_i - 1]$ that represents $\text{peg}_{i}$’s physical memory.

16) Given a submachine segment $D_j$ for submachine $S_j \in Q$, the partitionable SIMD/SPMD addressing constraints state that $D_j$ can be placed within the PEG memory maps for all $\text{peg}_{i} \in S_j$ only if there exists a base address, $\alpha_j$,.
for $0 \leq \alpha_j < (u_i - d)$ such that for all $peg_i \in S_j$, $U_j[\alpha_j \ldots \alpha_j + d - 1] = D_j[0 \ldots d - 1]$. The same $\alpha_j$ must be used for all $peg_i \in S_j$.

Fig. 4 is a graphical depiction of the four PEG memory maps ($U_j$'s) that correspond to the four PEG's associated with the running example machine $M$. These PEG memory maps contain the submachine segments specified in Fig. 3 for submachine set $Q'$. Each location represents a fixed-size block of memory, e.g., each location corresponds to 1K words. Additionally, unused portions of the PEG memory maps, up to the maximum required address (20), are represented with diagonal lines. Because each submachine segment has the same base address for all of the PEG memory maps that it includes (e.g., for submachine segment $D_3$, a base address of four is used by both $peg_2$ and $peg_3$), the set of PEG memory maps in Fig. 4 adheres to the partitionable SIMD/SPMD addressing constraints.

To provide an example of the relationship between the execution of a task and the PEG memory maps generated for it, consider a task $E'$ that is to execute on the running example machine $M$ and that draws its partitionings from the partitioning set $R'$. Let this task be defined as $E' = (\pi_0, \pi_1, \pi_2, \pi_3) = (P_0, P_1, P_2, P_3)$, where $P_0$, $P_1$, and $P_2$ are as previously defined. Figs. 5 and 6 illustrate the relationship between the execution of $E'$ and the PEG memory maps generated for $E'$.

Fig. 5 depicts a sample execution of $E'$. Within Fig. 5, a box labeled "$S_i\pi_j$" indicates that it represents the execution, in time, of submachine $S_i$ when it is operating in partitioning $\pi_j$. It is assumed that all submachines within a partitioning must complete execution of their corresponding subtasks before the machine can be reconfigured for the next partitioning in the task. Therefore, when a submachine has completed its execution within a specific partitioning, it must wait until all submachines within the partitioning complete executing as well. The cross-hatched areas in Fig. 5 represent submachine idle times. For example, Fig. 5 indicates that before the PEG's in submachine $S_1$ can proceed with partitioning $\pi_0$, these PEG's must remain idle until submachine $S_3$ completes in partitioning $\pi_3$.

Fig. 6 depicts a sample set of PEG memory maps for $E'$. Within Fig. 6, a box labeled "$D_i\pi_j$" represents the portion of the submachine segment $D_i$ executed during partitioning $\pi_j$. A dashed line in Fig. 6 indicates that a submachine segment was accessed during multiple partitionings; e.g., $S_1$ is part of $\pi_0$ and $\pi_2$, where $\pi_0$ and $\pi_2$ occur at different times, however all of $S_1$'s data and code for both $\pi_0$ and $\pi_2$ are stored in $D_1$. There is no correlation between the length of time a submachine executes (the height of a box in Fig. 5) and the number of locations required for a submachine's code/data (the height of a box in Fig. 6).

17) A concise set of PEG memory maps is a set of PEG
memory maps where for any segment $D_i$, corresponding to submachine $S_i \in Q$, that has base address $\alpha_i$ within the PEG memory maps, either $\alpha_i = 0$ or there exists a submachine $S_i$ such that the memory location $U_j[\alpha_i - 1]$ is occupied by another segment (associated with another submachine). Alternately stated, all segments within a concise set of PEG memory maps are bounded from below by either the bottom of the memory map or by a different segment.

18) A PEG memory map hole is any unused portion of a PEG memory map that is located between address 0 and at least one submachine segment base address.

19) A perfect set of PEG memory maps is a set of PEG memory maps that does not contain any PEG memory map holes.

The set of PEG memory maps shown in Fig. 4 is concise. However, this set is not perfect because there exist four PEG memory map holes (one in memory map $U_1$ over locations 2 to 3, one in memory map $U_2$ over locations 8 to 10, and two in memory map $U_3$ over locations 0 to 4 and 8 to 10). Methods for “packing” submachine segments and sets of submachine segments into a set of PEG memory maps are defined below.

20) A D-packing of some submachine segment $D_i$ for submachine $S_i \in Q$ consists of placing $D_i$ into the PEG memory maps associated with each PEG in $S_i$ at some base address $\alpha_i$, subject to the partitionable SIMD/SPMD addressing constraints and such that no other previously D-packed segments are overwritten.

21) A concise D-packing is a D-packing whose base address $\alpha_i$ is made as small as possible while still adhering to the constraints associated with a D-packing.

22) A Q-packing of a submachine set $Q$ consists of D-packing each submachine segment associated with a submachine in $Q$. If $S_i \in Q$, but $S_i \notin V$, then $d_i = 0$ (segment $D_i$ is empty).

Consider the set of PEG memory maps displayed in Fig. 4. Submachine segment $D_1$ is D-packed at location (base address) three over each memory map associated with PEG's in submachine $S_1 = \{peg_1, peg_2\}$. At the time when $D_1$ is to be D-packed, if $D_4$ is the only other submachine segment in the set of PEG memory maps and $D_4$ has already been D-packed at location zero, $D_1$ is concisely D-packed at location three. The set of PEG memory maps in Fig. 4 represents a Q-packing of the submachine set $Q$ corresponding to the running example machine $M$.

23) A concise Q-packing is a Q-packing that only uses concise D-packings and that assigns a temporal ordering to all of the concise D-packings. Let such an ordering be represented by a total ordering of $Q$ (a Q-pack ordering). Note that a concise Q-packing always results in a concise set of PEG memory maps.

24) A Q-pack ordering $T$ of a submachine set $Q$ is a total ordering of $Q$, i.e., $(Q, \leq)$, where if $S_i \in Q$ has segment $D_i$, $S_j \in Q$ has segment $D_j$, and $S_i \leq S_j$ within $T$, then $D_i$ is concisely D-packed before $D_j$ in time. Given a submachine segment $Q$, the only variation in concise Q-packings of $Q$ is in the choice of which Q-pack ordering to use.

25) A perfect Q-packing is one that results in a set of PEG memory maps that is perfect.

Continuing the running example, assume that the task $E'$ possesses submachine segments ($D_i$'s) with corresponding sizes ($d_i$'s) as displayed in Fig. 3. Let $T_1$ be the following hypothetical total ordering of $Q'$: $S_4 \leq S_1 \leq S_6 \leq S_3 \leq S_5 \leq S_7$. If $T_1$ is used as a Q-pack ordering, then a concise Q-packing of the submachine set $Q'$ results in the concise set of PEG memory maps shown in Fig. 4. Because there exist four PEG memory map holes as previously pointed out, the set of PEG memory maps in Fig. 4 is not perfect.

Now consider a specially chosen Q-pack ordering $T_2$ represented by the following total ordering of $Q'$: $S_6 \leq S_1 \leq S_5 \leq S_3 \leq S_4 \leq S_6 \leq S_7$. When $T_2$ is used, rather than $T_1$, a concise Q-packing results in the perfect set of PEG memory maps shown in Fig. 7. The largest memory requirement over all PEG's is 14 locations when $T_2$ is used as the Q-pack ordering, whereas when $T_1$ was used, the requirement was 20 locations.

A basic assumption of the memory model just presented is that it adheres to the partitionable SIMD/SPMD addressing constraints that state that all processors within the same submachine are required to use identical addresses to access corresponding data items allocated in each of their local memories. An alternative approach would be to lift this constraint. An example of this consequence is shown in Fig. 8, which is a modification of the memory maps in Fig.
4 (corresponding to the Q-pack ordering \( T_1 \)). In Fig. 8, fragmentation is reduced because submachine segment \( D_0 \) uses different base addresses with respect to different PEG’s. Such an alternate approach would require additional hardware and software for the support of both independent base registers and tracking multiple address spaces when accessing variables across different submachines. Using Fig. 8 as an example, if submachine \( S_2 \) (corresponding to \( D_0 \)) generated intermediate parallel results to be accessed at a later time by submachine \( S_3 \) (corresponding to \( D_3 \)), the single program being executed on \( S_3 \) would need to access the intermediate results at different addresses with respect to \( \text{PEG}_2 \) and \( \text{PEG}_3 \). This alternative approach is not considered here for both theoretical and practical reasons. Theoretically, it would make the memory map problem unintertesting; and practically, because large-scale machines are being addressed, the hardware and software overhead discussed above could become significant.

Another basic assumption of the memory model used is that submachine segments (\( D \)'s) cannot be decomposed so as to fill existing PEG memory map holes. Fig. 9 is a modification of Fig. 4 (corresponding to the Q-pack ordering \( T_1 \)) in which \( D_T \) is such a decomposed submachine segment. Reasons for maintaining this assumption are 1) packing data arrays with “memory gaps” destroys sequential memory access, 2) packing code segments with “memory gaps” requires the insertion of direct branches, and 3) submachine segments could no longer be separately compiled.

The following section uses the above model to derive the necessary and sufficient conditions on a partitionable SIMD/SPMD machine that guarantee to achieve a perfect Q-packing for all tasks the machine executes. The results put forth in the next section demonstrate that for a class of partitionable SIMD/SPMD machines, all fragmentation can be avoided without the need for base registers or decomposing submachine segments.

III. PERFECT MEMORY MAPS FOR PARTITIONABLE SIMD/SPMD MACHINES

This section treats the problem of Q-packing a set of submachine segments obtained from an arbitrary parallel task so that the resulting set of PEG memory maps created does not suffer from any fragmentation. The results presented in this section are used to derive necessary and sufficient conditions on a partitionable machine for being able to always obtain a perfect Q-packing. These results are important because if perfect Q-packings are not obtainable, then the memory map creation problem for partitionable SIMD/SPMD machine tasks reverts into a variation of the bin packing problem.

Consider the creation of PEG memory maps at compile-time for a partitionable SIMD/SPMD machine task. It is possible for a compiler to determine what submachine segments will be needed on which PEG’s so that all segments are not sent to all PEG’s. A particular PEG memory map generated for a partitionable SIMD/SPMD machine task will consist of the submachine segments from those submachines that incorporate the PEG.

The proposed method for assimilating a set of submachine segments generated from a parallel task into a set of PEG memory maps is to use a concise Q-packing. The motivation behind this is provided by the following theorem.

Theorem 1: Assume that a given partitionable machine \( M \) has a corresponding submachine set \( Q \). Given a set of submachine segments generated from a particular partitionable SIMD/SPMD task, if there exists a Q-packing that is perfect, then there exists a concise Q-packing that is also perfect.

Proof: By contradiction. Assume that there exists a perfect Q-packing and that all concise Q-packings are not perfect. It is shown that there exists a Q-packing for a concise Q-packing that results in the same perfect set of PEG memory maps that are generated by the perfect Q-packing that is assumed to exist. Let the Q-packing order be a topological sort over the precedences that show which submachine segments are “placed” above other segments in space within the perfect Q-packing assumed to exist. Specifically, for all \( S_x, S_y \in Q \), let \( \alpha_x \) and \( \alpha_y \) be the base addresses of \( S_x \)'s segment and \( S_y \)'s segment, respectively, within the perfect Q-packing that exists. If \( (S_x \cap S_y) \neq \emptyset \) and if \( \alpha_x < \alpha_y \), then \( S_x \) must precede \( S_y \) within the Q-packing ordering. A concise Q-packing using this Q-packing will result in a perfect set of PEG memory maps.

Due to the fact that the only variation allowed within a concise Q-packing is its Q-packing ordering, the number of distinct concise Q-packings for a submachine set \( Q \) must be \(|Q|!\). Therefore, Theorem 1 is significant because it says that if there exists a way to perfectly “pack” a set of submachine segments into a set of PEG memory maps, then one of a possible \(|Q|!\) concise Q-packings must exist that can also do it perfectly. The following theorem indicates why perfect Q-packings are desirable.

Theorem 2: For each possible Q-pack ordering \( T_1 \) for a given partitionable SIMD/SPMD task, let \( \mu_i \) be the maximum memory size required for any PEG. A perfect Q-packing yields the minimum \( \mu_i \) over all possible Q-pack orderings.

Proof: Given that a perfect Q-packing cannot contain any PEG memory map holes, if a Q-packing ordering \( T_1 \) results in a perfect Q-packing then all unused memory locations within any particular PEG memory map must exist after the last D-packed submachine segment in space within the PEG memory.
map (e.g., see Fig. 7). This implies that $\mu_1$ must be a minimum over all the possible $\mu_1$’s.

The following theorem provides a necessary condition on the fact that a partitionable machine’s submachine set is nested. This result is employed in Section IV.

**Theorem 3:** Assume that a given partitionable machine $M$ has a corresponding submachine set $Q$ and let $H$ be the Hasse diagram of the partial ordering $(Q, \supseteq)$. If $Q$ is nested then $H$ is a forest of one or more trees.

**Proof:** By contradiction. Assume that $Q$ is nested and that $H$ is not a forest of one or more trees. Because $H$ is a DAG and not a forest, $H$ must contain a node $S \in Q$ that has more than one immediate predecessor within $(Q, \supseteq)$. Let two of these immediate predecessors be $S_x \in Q$ and $S_y \in Q$. For example, in Fig. 1, the Hasse diagram is a DAG and not a tree because the node $S_3 = \{\text{peg}_1\}$ has exactly two immediate predecessors, namely, $S_1 = \{\text{peg}_5, \text{peg}_6\}$ and $S_2 = \{\text{peg}_1, \text{peg}_2\}$. Due to the fact that $S \not\subseteq S_x$ and $S \not\subseteq S_y$, it must be the case that $(S_x \cap S_y) \neq \emptyset$. Because $Q$ is nested, either $S_x \subset S_y$ or $S_y \subset S_x$. Without loss of generality, assume that $S_x \subset S_y$; therefore, there must be a path from $S_y$ to $S_x$ within $H$. Because there is an edge from $S_y$ to $S$, this contradicts the fact that $S_y$ is an immediate predecessor of $S$ within $H$.

The remaining group of theorems derive the conditions on a partitionable machine $M$ that are necessary and sufficient for being able to perfectly $Q$-pack the submachine segments from any parallel task executed on $M$. Corollary 2 is the main result from this section.

**Theorem 4:** Assume that a given partitionable machine $M$ has a corresponding submachine set $Q$ and let $T$ be a $Q$-pack ordering of $Q$. If $Q$ is nested and if $T$ is a topological sort of the partial ordering $(Q, \supseteq)$, then all partitionable SIMD/SPMD tasks executed on $M$ that are concisely $Q$-packed by $T$ will have perfect PEG memory maps.

**Proof:** By contradiction. Assume that $Q$ is nested, that $T$ is a topological sort of $(Q, \supseteq)$, and that there exists a partitionable SIMD/SPMD task executed on $M$ that when concisely $Q$-packed by $T$ will not have perfect PEG memory maps. Therefore, no matter what $Q$-pack ordering is used to concisely $Q$-pack the submachine segments of this task within $M$’s PEG memory maps, there will always at least one PEG memory map hole. This implies that there will always exist a submachine segment $D_x$ (corresponding to $S_x \in Q$) with base address $\alpha_x > 0$ that bounds the PEG memory map hole from above in space. For example, in Fig. 4, the peg$_3$ memory map hole over locations 0 to 4 is bounded from above by submachine segment $D_x$. By definition of a concise $Q$-packing, $D_x$ was concisely $D$-packed. This implies that there exists another submachine segment $D_y$ (corresponding to $S_y \in Q$) on which $D_x$ was “placed.” Referring back to Fig. 4, if submachine segment $D_3$ corresponds to submachine segment $D_x$, then segment $D_8$ corresponds to segment $D_y$. Due to the PEG memory map hole, there exists a PEG memory map which contains $D_x$ and does not contain $D_y$ (in Fig. 4 where $D_3$ and $D_8$ correspond to $D_x$ and $D_y$, respectively, this PEG memory map is the one for peg$_3$). Coupling this with the facts that $D_x$ was “placed on” $D_y$ and that $Q$ is nested implies that $S_y \subset S_x$. Therefore, because $T$ is a topological sort of $(Q, \supseteq)$, $D_x$ must be concisely $D$-packed before $D_y$ in time. However, this leads to a contradiction because $D_x$ was concisely $D$-packed “on” $D_y$ which implies that $D_y$ was concisely $D$-packed before $D_x$ in time.

**Theorem 5:** Assume that a given partitionable machine $M$ has a corresponding submachine set $Q$ and let $T$ be a $Q$-pack ordering of $Q$. If all partitionable SIMD/SPMD tasks executed on $M$ that are concisely $Q$-packed by $T$ will have perfect PEG memory maps, then $T$ is a topological sort of the partial ordering $(Q, \supseteq)$.

**Proof:** By contradiction. Assume that all partitionable SIMD/SPMD tasks executed on $M$ that are concisely $Q$-packed by $T$ will have perfect PEG memory maps and that $T$ is not a topological sort of $(Q, \supseteq)$. Choose a partitionable SIMD/SPMD task that operates solely on two submachines, i.e., task submachine set $V = \{S_x, S_y\} \subseteq Q$, where $S_x \not\subseteq S_y$. Because $T$ is not a topological sort of $(Q, \supseteq)$, $T$ must order $S_x$ before $S_y$. This ensures that $S_y$’s submachine segment is concisely $D$-packed after $S_x$’s submachine segment in time. Due to the fact that $S_x \subset S_y$ implies that $(S_x \cap S_y) \neq \emptyset$, $S_y$’s segment is concisely $D$-packed after $S_x$’s segment in space. Because $S_x$ and $S_y$ are the only submachines that have segments to concisely $D$-pack, PEG memory map holes are created within all PEG memory maps for PEG’s in the nonnull set $S_y - S_x$, contradicting the assumption that the memory maps are perfect. For example in Fig. 4, let $S_x = \{\text{peg}_2\}$, $S_y = \{\text{peg}_3, \text{peg}_4\}$, $D_x = D_6$, and $D_y = D_5$ (ignore all other submachine segments in the figure). Then there will be a PEG memory map hole (over locations 0 to 4) for PEG peg$_3$ $= S_y - S_x$.

**Theorem 6:** Assume that a given partitionable machine $M$ has a corresponding submachine set $Q$ and let $T$ be a $Q$-pack ordering of $Q$. If all partitionable SIMD/SPMD tasks executed on $M$ that are concisely $Q$-packed by $T$ will have perfect PEG memory maps, then $Q$ is nested.

**Proof:** By contradiction. Assume that all partitionable SIMD/SPMD tasks executed on $M$ that are concisely $Q$-packed by $T$ will have perfect PEG memory maps and that $Q$ is not nested. Choose a partitionable SIMD/SPMD task that operates solely on two submachines, $V = \{S_x, S_y\} \subseteq Q$, where $(S_x \cap S_y) \neq \emptyset$, $S_x$ is not a subset of $S_y$, and $S_y$ is not a subset of $S_x$. These circumstances directly guarantee that $Q$ is not nested. By Theorem 5, because all partitionable SIMD/SPMD tasks executed on $M$ can have their submachine segments concisely $Q$-packed to form perfect PEG memory maps, $T$ is a topological sort of $(Q, \supseteq)$. Due to the fact that $S_x$ is not a subset of $S_y$ and vice versa, $T$ can order $S_x$ before or after $S_y$; without loss of generality, assume that $T$ orders $S_x$ before $S_y$. This ensures that $S_y$’s submachine segment is concisely $D$-packed after $S_x$’s submachine segment in time. Further, because $(S_x \cap S_y) \neq \emptyset$, $S_y$’s segment is concisely $D$-packed after $S_x$’s segment in space. Because $S_x$ and $S_y$ are the only submachines that have segments to concisely $D$-pack, PEG memory map holes are created within all PEG memory maps for PEG’s in the nonnull set $S_y - S_x$, contradicting the assumption that the memory maps are perfect. For example, consider Fig. 10 which assumes that $S_x = \{\text{peg}_1, \text{peg}_2\}$ and
\( S_p = \{ \text{peg}_2, \text{peg}_3 \} \). There is a PEG memory map hole over locations 0 to \( d_x \) for PEG's within the set \( S_p - S_x = \{ \text{peg}_3 \} \).

**Corollary 1**: (Converse of Theorem 4) Assume that a given partitionable machine \( M \) has a corresponding submachine set \( Q \) and let \( T \) be a \( Q \)-pack ordering of \( Q \). If all partitionable SIMD/SPMD tasks executed on \( M \) that are concisely \( Q \)-packed by \( T \) will have perfect PEG memory maps, then \( Q \) is nested and \( T \) is a topological sort of the partial ordering \((Q, \supseteq)\).

**Proof**: Follows from Theorems 5 and 6.

**Corollary 2**: Assume that a given partitionable machine \( M \) has a corresponding submachine set \( Q \) and let \( T \) be a \( Q \)-pack ordering of \( Q \). All partitionable SIMD/SPMD tasks executed on \( M \) that are concisely \( Q \)-packed by \( T \) will have perfect PEG memory maps if and only if \( Q \) is nested and \( T \) is a topological sort of the partial ordering \((Q, \supseteq)\).

**Proof**: Follows from Theorem 4 and Corollary 1.

As an example of Corollary 2, \( Q', \) shown in Fig. 2, is nested, and the \( Q \)-pack ordering \( T_1 \) defined in Section II is a topological sort of the partial ordering \((Q', \supseteq)\). The resulting perfect \( Q \)-packing was shown in Fig. 7. The \( Q \)-pack ordering \( T_1 \), defined in Section II, for the nested \( Q' \) is not a topological sort (\( S_1 \) is not a subset of \( S_2 \)), and the resulting \( Q \)-packing shown in Fig. 4 is not perfect. The \( Q \) shown in Fig. 1 is not nested, and for the example in Fig. 10, even though the \( Q \)-packing is representable by a topological sort, the \( Q \)-packing is not perfect.

If a partitionable machine supports a submachine set \( Q \) that is nested, the only constraint on the \( Q \)-pack ordering that ensures a perfect \( Q \)-packing is that it be a topological sort of the partial ordering \((Q, \supseteq)\). As long as \((Q, \supseteq)\) is not a total ordering, many topological sorts of \((Q, \supseteq)\) will exist. The process of determining such a topological sort at compile-time is straightforward. For example, given that \( Q \) is nested, a breadth-first traversal of the Hasse diagram of the partial ordering \((Q, \supseteq)\) (which by Theorem 3 is a forest of one or more trees) is equivalent to a topological sort of \((Q, \supseteq)\). Therefore, provided that \( Q \) is nested, the complexity of computing a viable \( Q \)-pack ordering is \( O(|Q|) \).

In some instances, Corollary 2 can be invoked on the SIMD code segments for each submachine used in a partitionable SIMD task. If each PEG has its own control unit that broadcasts instructions (as in PASM), the partitionable SIMD/SPMD addressing constraints can be applied to the control unit memory maps that contain the SIMD code to be broadcast. On such a partitionable SIMD system (i.e., multiple-SIMD system), Corollary 2 could be used so that the submachine code segments could be concisely \( Q \)-packed to form perfect memory maps.

By adhering to certain constraints, the results derived in this section can also be applied to nonpartitionable machines capable of SIMD and/or SPMD operation. Assume the \( N \) PE's in a machine are numbered from 0 to \( N - 1 \). If PE's are referenced by their unique addresses (or numbers) within a nonpartitionable SIMD and/or SPMD program, a compiler could be used to determine which sets of PE's within the nonpartitionable machine can possibly access particular parallel variables. (These PE sets are similar to submachines in a partitionable machine.) Let \( \Sigma_{v_i} \) be the set of PE's within a nonpartitionable SIMD and/or SPMD program that have access to parallel variable \( v_i \). Possessing such information would allow unique memory maps to be generated for each PE. Specifically, a PE's unique memory map would not contain storage space for parallel variables that it could not access. Similar to the partitionable SIMD/SPMD addressing constraints, assume that a parallel variable must be located at the same address within each memory map corresponding to a PE that has access to the parallel variable. Given a nonpartitionable SIMD and/or SPMD program, let \( \Phi \) be the set containing all \( \Sigma_{v_i} \)'s corresponding to parallel variables \( v_i \)'s used in the program. Therefore, Corollary 2 can be applied to nonpartitionable SIMD and/or SPMD machines only if \( \Phi \) is nested (i.e., if \( \Sigma_{v_i}, \text{and} \ v_i \in \Phi \), then \( \Sigma_{v_i} \cap \Sigma_{v_j} \neq \emptyset \)) implies that either \( \Sigma_{v_i} \subset \Sigma_{v_j} \) or \( \Sigma_{v_j} \subset \Sigma_{v_i} \).

**IV. PERFECT MEMORY MAPS BASED ON SPECIFIC INTERCONNECTION NETWORKS**

If the partitioning capability of partitionable SIMD/SPMD machine \( M \) satisfies Corollary 2, the PEG memory maps for any parallel task executed on \( M \) can be perfectly \( Q \)-packed. Because, for the most part, a machine's partitioning capability is based on the partitioning capability of the interconnection network it employs, this section will examine two classes of common partitionable networks. This section also examines what can be done if perfect memory maps are desired on a partitionable machine that does not satisfy Corollary 2 (i.e., \( Q \) is not nested). It is concerning this point that a distinction must be made between a machine's partitioning capability and the partitioning capability provided for the programmer. These two do not necessarily have to be the same. Specifically, this section looks at the case where the submachine set \( Q \) is not nested, and by forcing the programmer to utilize submachines from a \emph{reduced submachine set} \( Q' \subset Q \), it is possible for the new submachine set \( Q' \) to be nested. This method can be used successfully with partitionable machines employing single-stage cube (hypercube) [30], [32] or multistage cube [32], [33] type interconnection networks. It will also be shown that the PM2I [30], [32] and data manipulator family [12], [32] interconnection networks satisfy Corollary 2. Without loss of
generality, only the single-stage cube and PM2I networks will be studied in depth in this section, because their partitioning properties are the same as those of the multistage cube and data manipulator family, respectively [31].

Interconnection networks provide a means for transferring data among the PE’s of a parallel machine. A significant feature supported by many interconnection networks is the ability to partition themselves into independent subnetworks that each maintain complete network functionality [31], [32]. Given a machine with \( N = 2^m \) PE’s and given that the PE’s are addressed \( 0 \) through \( N - 1 \), the PM2I interconnection network is based on the following connections:

\[
\text{PM2}_{+i}(P) = P + 2^i \mod N
\]

and

\[
\text{PM2}_{-i}(P) = P - 2^i \mod N \quad \text{for} \quad 0 \leq i < m
\]

where \( P \) is the address of an arbitrary PE. The connections for the single-stage cube (hypercube) interconnection network are

\[
\text{cube}_i(P) = P_{n-1} \cdots P_{i+1}P_iP_{i-1} \cdots P_0
\]

where \( P_{n-1} \cdots P_{i+1}P_iP_{i-1} \cdots P_0 \) is the binary address of an arbitrary PE, and \( P_i \) is the one complement of bit \( P_i \). Thus, with the PM2I network, PE \( P \) is connected to the \( 2m \) PE’s \( \text{PM2}_{+i}(P) \), \( 0 \leq i < m \), and with the single-stage cube network, PE \( P \) is connected to the \( m \) PE’s \( \text{cube}_i(P) \), \( 0 \leq i < m \). Of the machines mentioned in Section I, the Connection Machine, iPSC Hypercube, and NCube system employ single-stage cube interconnection networks.

All submachines created from the partitioning of single-stage cube and PM2I networks can be represented by using PE address masks for PEG’s rather than for PE’s. Given a machine with \( N = 2^m \) PE’s addressed 0 to \( N - 1 \), a PE address mask [30], [32] is an \( m \) position mask, \( \{m_{-1} \cdots m_0\} \), used to specify a set of PE’s, where each position of the mask corresponds to a bit position in the PE addresses. Each position of the mask is either a 0, 1, or \( X \) (“don’t care”). For example, the set of even-numbered PE’s is represented by the mask \( [X^{m-1}0] \), where the notation \( X^m \) is a string of \( m - 1 \) \( X \)’s. This section assumes that a partitionable machine \( M \) has \( B = 2^k \) PEG’s addressed 0 to \( B - 1 \); therefore, a submachine mask is a \( b \)-position mask, \( \{0 \cdots 0\} \), used to specify sets of PEG’s instead of sets of PE’s, where each position of the mask corresponds to a bit position in the PEG addresses. Based on the partitioning rules for both single-stage cube and PM2I interconnection networks [32], submachine masks can represent all distinct submachines that can be generated by partitioning either network.

The PM2I interconnection network is partitioned by starting with the submachine mask \( [X^b] \) (representing the submachine containing all of the machine’s \( B = 2^b \) PEG’s) and progressively “instantiating” mask positions that contain \( X \)’s from the least significant position to the most significant position to create additional submachine masks. The instantiation of a submachine mask at a mask position containing an \( X \) consists of creating two new submachine masks: one mask that is the same as the original except for a 1 in the \( X \)’s mask position and another mask that is the same as the original except for a 0 in the \( X \)’s mask position. Let \( Q_{PM2I} \) be the submachine set obtained from PM2I partitionings and let \( H_{PM2I} \) be the Hasse diagram of all allowable submachines. \( H_{PM2I} \) represents the partial ordering \( (Q_{PM2I}, \leq) \). Assuming that submachine \( [X^b] \) is at zero depth within \( H_{PM2I} \), the two submachines \( [X^{b-1}] \) (all odd-numbered PEG’s) and \( [X^{b-1}] \) (all even-numbered PEG’s) obtained by instantiating submachine \( [X^b] \) at mask position 0 are each at depth one within \( H_{PM2I} \). Similarly, by instantiating both submachines \( [X^{b-1}] \) and \( [X^{b-1}] \) at mask position 1, the four resulting submachines \([X^{b-1}2],[X^{b-1}2],[X^{b-2}0],[X^{b-2}0],[X^{b-2}01],[X^{b-2}01]\) are each at depth two within \( H_{PM2I} \). In general, by instantiating at mask position \( i \) \( 0 \leq i < b \) each of the \( 2^i \) submachines at depth \( i \) within \( H_{PM2I} \), \( 2^{i+1} \) new submachines at depth \( i + 1 \) within \( H_{PM2I} \) are created. By continuing this process through depth \( b - 1 \), it is seen that \( H_{PM2I} \) is a complete binary tree with height \( b \); therefore,

\[
|Q_{PM2I}| = 2^{b+1} - 1
\]

Fig. 11 shows, for \( b = 4 \), the submachines within \( Q_{PM2I} \) represented by their submachine masks and the Hasse diagram \( H_{PM2I} \) incorporating all of those submachines. It can be seen that \( Q_{PM2I} \), for \( b = 4 \), is nested. By Theorem 3, a consequence of this is that \( H_{PM2I} \) is a tree.

In the PM2I instantiation process for constructing \( H_{PM2I} \): 1) all submachine masks at depth \( i \) within \( H_{PM2I} \) are instantiated at mask position \( i \), and 2) from depth zero to depth \( b - 1 \), instantiations are ordered from mask position 0 (least significant) to mask position \( b - 1 \) (most significant). The PM2I network’s partitioning rules require 1) and 2) [32] because any PM2I submachine \( S_{PM2I} \) with \( K = 2^b \) PEG’s must be represented with a submachine mask whose least significant \( b - k \) mask positions are either 0’s or 1’s and whose most significant \( k \) mask positions are \( X \)’s. This can be seen by the fact that \( S_{PM2I} \) is located at depth \( b - k \) within \( H_{PM2I} \). Therefore, the addresses of all PEG’s in \( S_{PM2I} \) agree in their low-order \( b - k \) bit positions. For example, in Fig. 11, for \( b = 2 \) and \( k = 1 \), one \( S_{PM2I} \) with \( K = 2 \) PEG’s is \([X1] \), at depth one, consisting of \([P_{01}, P_{01}] \).

The process of creating PM2I submachine masks by instantiation simply starts with the set of all PEG’s and progressively divides this set and resulting sets in half, therefore, \( Q_{PM2I} \) is guaranteed to be nested. Thus, by Corollary 2, given that a partitionable machine is using a PM2I interconnection network, all partitionable SIMD/SMPD tasks executed on the machine can have perfect memory maps.

When the single-stage cube interconnection network is considered, Corollary 2 no longer applies. The reason is that the partitioning capability of the single-stage cube network is much more flexible than that of the PM2I network. Specifically, restrictions 1) and 2) on the instantiation of submachine masks when forming PM2I submachines are not necessary during the formation of single-stage cube submachines. The total flexibility during the submachine mask instantiation process obtained when these restrictions are lifted implies that, for a single-stage cube network, there exists a bijection between the set of all possible distinct \( b \)-position submachine masks and
the set of all distinct single-stage cube submachines, \( Q_{\text{cube}} \). Therefore, \( |Q_{\text{cube}}| = 3^b \), which is considerably larger than \( |Q_{\text{PM2I}}| \).

Let \( H_{\text{cube}} \) be the Hasse diagram representing the partial ordering \( (Q_{\text{cube}}, \supseteq) \). Fig. 12 shows, for \( B = 4 \), the submachines within \( Q_{\text{cube}} \) represented by their submachine masks and \( H_{\text{cube}} \) incorporating all of those submachines. Using the single-stage cube network, a submachine consisting of a single PEG \([x_{b-1} \cdots x_0]\) at depth \( b \) can be formed from any of \( b \) nodes at depth \( b-1 \). Specifically, the \( b \) nodes are those obtained from replacing one position of \([x_{b-1} \cdots x_0]\) with an \( X \). For example, in Fig. 12, \([10]\) can be instantiated from \([X0]\) and \([1X]\). This is sufficient to show that \( H_{\text{cube}} \) is a DAG and not a tree, so by Theorem 3 \( Q_{\text{cube}} \) is not nested. Consequently, Corollary 2 cannot be directly applied to single-stage cube interconnection networks.

A PM2I network’s submachine set \( Q_{\text{PM2I}} \) is nested because all of the submachines in \( Q_{\text{PM2I}} \) (except \([X^b]\)) can be generated via the submachine mask instantiation process described above and because the instantiation of a submachine mask containing an \( X \) simply divides the original submachine into two new submachines of equal size. Therefore, the fact that \( Q_{\text{PM2I}} \) is nested does not depend on the submachine mask instantiation restrictions 1) and 2) described above. As previously noted, these restrictions are also lifted during the submachine mask instantiation process for partitioning single-stage cube machines into submachines. Consequently, a different restriction on the submachine mask instantiation process for single-stage cube networks must be determined so that reduced submachine sets (defined to be nested) can be constructed.

Given a reduced submachine set \( Q'_{\text{cube}} \), let \( H'_{\text{cube}} \) be the corresponding Hasse diagram of the partial ordering \( (Q'_{\text{cube}}, \supseteq) \). By Theorem 3, the fact that \( H'_{\text{cube}} \) is a single tree is necessary for \( Q'_{\text{cube}} \) to be nested. Because the instantiation of a submachine mask simply subdivides the submachine into two equal parts, if a Hasse diagram generated by the submachine mask instantiation process is a tree, the submachine set corresponding to the Hasse diagram is nested. Consequently, the fact that \( H'_{\text{cube}} \) is a tree is necessary and sufficient for \( Q'_{\text{cube}} \) to be nested. Therefore, if the submachine mask instantiation process for single-stage cube networks is restricted to generating a Hasse diagram \( H'_{\text{cube}} \) that is a complete binary tree of height \( b \) (instead of generating a DAG that is not a tree) whose root is the submachine containing all \( B = 2^b \) PEG’s and whose leaves are the \( B \) distinct submachines containing a single PEG, the reduced submachine set \( Q'_{\text{cube}} \) is nested and is maximal (i.e., its cardinality is as large as possible). This restriction obviates the single-stage cube network’s partitioning rules [32] because any single-stage cube submachine \( S'_{\text{cube}} \) with \( K = 2^b \) PEG’s must be represented with a submachine mask where any \( b - k \) mask positions are either 0’s or 1’s and \( k \) mask positions are \( X \)’s. This can be seen by the fact that \( S'_{\text{cube}} \) is located at depth \( b - k \) within \( H_{\text{cube}} \). Therefore, the addresses of all PEG’s in \( S'_{\text{cube}} \) agree in exactly \( b - k \) bit positions.

The difference between generating a nested \( Q_{\text{PM2I}} \) for the PM2I network and generating a nested \( Q'_{\text{cube}} \) for the single-stage cube network is that while there are many different nested \( Q_{\text{cube}} \)’s (that are enumerated below), there is only a single nested \( Q_{\text{PM2I}} \). For the PM2I network, because the submachine mask instantiation restrictions 1) and 2) apply, there is only one way to choose the submachines located at a specific depth within \( H_{\text{PM2I}} \). With the single-stage cube network, because the only restriction is that the resulting Hasse diagram be a complete binary tree, there are multiple ways to choose the submachines located at a specific depth within \( H'_{\text{cube}} \). For example, at depth zero (root level) of an \( H'_{\text{cube}} \) any one of the \( b \) mask positions that contain an \( X \) can be used for instantiation. If mask position \( i \) for \( 0 \leq i < b \) is used for instantiation, two submachines at depth one within \( H'_{\text{cube}} \) are created (one with a 0 in mask position \( i \) and one with a 1 in mask position \( i \), while all their other mask positions are \( X \)’s). Subsequently, both of these resulting two submachine masks can each be instantiated using one of their \( b - 1 \) mask positions that contain an \( X \). The choice of which mask position to use for the instantiation can be done independently with respect to both submachine masks and results in the creation of four submachines at depth two within \( H'_{\text{cube}} \). This submachine mask instantiation process continues down to depth \( b - 1 \) within \( H'_{\text{cube}} \).

Consider the enumeration of all possible distinct \( Q_{\text{cube}} \)’s. At
depth zero within an $H_{cubex}$, there are $b$ different instantiation choices for the root submachine as previously discussed. At depth one, there are $b - 1$ different instantiation choices for each of the two submachines at that depth, with the two instantiations being independent of one another, giving $(b - 1)^2$ total choices. At depth two, there are $b - 2$ distinct instantiation choices for each of the four submachines at that depth, with the four instantiations being independent of one another, giving $(b - 2)^2$ total choices. In general, at depth $i$ within $H_{cubex}$, there are $b - i$ instantiation choices for each of the $2^i$ submachines at that depth, with the $2^i$ instantiations being independent of one another, giving $(b - i)^2$ total choices. Thus, the overall number of nested $Q_{cubex}$'s that can be created is $\prod_{i=0}^{k-1} (b - i)^2$.

As an example, for $B = 4$ (and $b = 2$) the number of nested $Q_{cubex}$'s is $\prod_{i=0}^{1} (2 - i)^2 = 2$. The Hasse diagram for one of these $Q_{cubex}$'s is shown in Fig. 11 (which is also $H_{PM2I}$ for $B = 4$) and the other is depicted in Fig. 13.

Thus, there are $\prod_{i=0}^{k-1} (b - i)^2$ different choices for $Q_{cubex}$ that are nested. By Corollary 2, given that a partitionable machine is using one of these $Q_{cubex}$'s, all partitionable SIMD/SPMD tasks executed on the machine can have perfect memory maps.

These results also apply to the multistage version of the PM2I network. The augmented data manipulator (ADM) network [21], [32] has $N = 2^m$ inputs and outputs and $m$ stages where each stage consists of $N$ switching elements (nodes) and $3N$ output links. There is also an $(m + 1)$st column of network output nodes. Fig. 14 illustrates an ADM network topology for $N = 8$. The inverse ADM (IADM) [20] and gamma [26] networks have topologies with similar partitioning properties. The connections in this network are based on the PM2I interconnection functions defined earlier in this section. Specifically, each node in stage $i$, $0 \leq i < m$, can be independently set to straight across, $-2^m$ modulo $N$, or $+2^m$ modulo $N$. The stages are ordered $m - 1$ to 0 from input to output. Because the partitionability of the ADM network is in direct correspondence with the partitionability of the PM2I network [31], [32], Corollary 2 can be applied to all machines employing an ADM, IADM, or gamma network.

The multistage version of the cube network, represented by the generalized-cube network topology [32], [33], has $N = 2^m$ inputs and outputs and $m$ stages where each stage consists of $N$ links and $N/2$ interchange boxes. The stages are ordered $m - 1$ to 0 from input to output. An interchange box has two inputs and two outputs and can be set to either map the outputs straight across from the inputs or map the outputs as a swap of the inputs. Fig. 15 shows a multistage cube network for $N = 8$. Other networks in this family include the baseline [39], indirect binary n-cube [27], multistage shuffle-exchange [37], omega [18], and SW-banyan (with $S = F = 2$) [19]. The connections in this network are based on the cube interconnection functions. The link labels of a stage $i$ interchange box differ in the $i$th bit position, i.e., stage $i$ of the generalized-cube network topology contains the cube, interconnection function. Partitioning a generalized-cube network is essentially the same as for a single-stage cube network. Once again it is possible to generate a $Q$ that is not nested (e.g., for Fig. 15, if stage 0 is set to straight, 0, 2, 4, and 6 form a submachine, while if stage 2 were set to straight instead, 0, 1, 2, and 3 form a submachine). Due to the correspondence with the single-stage cube network's partitionability, Corollary 2 cannot be applied to machines that employ a multistage cube network. However, by restricting the possible submachines to those in a $Q_{cubex}$ each of which can be supported by an independent multistage subnetwork [31], [32], Corollary 2 can then be applied. Of the machines mentioned in Section I, the Butterfly, PASM, and RP3 employ multistage cube interconnection networks.

V. CONCLUSION

Data layout is an important part of the compilation process. This work provides a model for creating fragment-free (perfect) memory maps on partitionable SIMD/SPMD parallel processing systems. Specifically, it has been shown what machine partitioning capabilities are necessary and sufficient for always being able to create perfect PEG memory maps for partitionable SIMD/SPMD tasks. These conditions are dependent on the set of all distinct submachines that can be generated during the partitioning process and on the order in which the submachine data segments are packed into the PEG memory maps. As discussed, the results can be applied to a variety of parallel architecture types.

In summary, as large-scale parallel processing systems become practical, it is important to understand the various software issues that will be associated with their efficient usage. Here, a model of memory maps for partitionable SIMD and/or SPMD machines was developed, and properties of these memory maps were derived.

ACKNOWLEDGMENT

The authors acknowledge many useful discussions with S.-D. Kim, C. Krauskopf, and C. Warner.
REFERENCES


Mark A. Nichols (S'86-M'91) received the B.S. degree with a triple major of electrical engineering, computer engineering, and mathematics (computer science) from Carnegie-Mellon University, Pittsburgh, PA, in 1985, the M.S.E.E. degree from the Georgia Institute of Technology, Atlanta, in 1986, and the Ph.D. degree in electrical engineering from Purdue University, West Lafayette, IN, in 1991. He is currently with NCR, San Diego, CA. While at Purdue, he was employed as the project leader for the design and implementation of a parallel reconfigurable PASM parallel processing system prototype. His research interests include parallel language/compiler design, parallel architecture modeling, and interconnection networks. Dr. Nichols is a member of the IEEE Computer Society and the Association for Computing Machinery.

Howard Jay Siegel (M'77 SM'82 F'90) received two B.S. degrees from the Massachusetts Institute of Technology, Cambridge, and the M.A., M.S.E., and Ph.D. degrees from Princeton University, Princeton, NJ. He is a Professor and Coordinator of the Parallel Processing Laboratory in the School of Electrical Engineering, Purdue University. He has coauthored over 150 technical papers, coedited four volumes, and written one book. He does consulting, has prepared two "continuing education" eleven-hour video-tape courses, and has given tutorials for various organizations in the U.S.A., Europe, and Israel, all in the area of parallel computing. His current research focuses on interconnection networks and the use and design of the unique PASM reconfigurable parallel computer system (a prototype of which is supporting active experimentation).

Dr. Siegel has served as Program Co-Chair of the 1983 International Conference on Parallel Processing, and as General Chair of the 3rd International Conference on Distributed Computing Systems and the 15th Annual International Symposium on Computer Architecture. He is Coeditor-in-Chief of the Journal of Parallel and Distributed Computing. He is a member of theEta Kappa Nu and Sigma Xi honorary societies.

Henry G. Dietz received the B.S., M.S., and Ph.D. degrees in computer science from the Polytechnic Institute of New York. In 1986, he joined the faculty of Purdue University where he is an Assistant Professor of Electrical Engineering (Computer Engineering). He has coauthored over 40 technical papers primarily in compiler optimization/parallelization and parallel architecture. He founded and leads CARP, the Compiler-oriented Architecture Research group at Purdue. Other current research activities include the development of FCCTS, the Purdue Compiler-Construction Tool Set, a collection of public domain software tools for construction of optimizing/parallelizing compilers.
Russell W. Quong (M'88) received the B.S. degree in electrical engineering from the California Institute of Technology, Pasadena, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Palo Alto, CA. He is an Assistant Professor in the School of Electrical Engineering, Purdue University. His research interests include hardware and system software design, programming environments, and analysis of algorithms.

Dr. Quong is a member of the Association for Computing Machinery and the IEEE Computer Society.

Wayne G. Nation (S'84-M'91) received the B.S.E.E. degree from the University of Kentucky, Lexington, in 1985, and the M.S.E.E. and Ph.D. degrees from the School of Electrical Engineering, Purdue University, West Lafayette, IN, in 1986 and 1991, respectively.

He is currently with IBM, Endicott, NY. While at Purdue, he was employed as the Parallel Processing Laboratory Manager. His research interests include interconnection networks, parallel processing, and parallel processor prototyping.

Dr. Nation is a member of Eta Kappa Nu, Tau Beta Pi, the IEEE Computer Society, and the Association for Computing Machinery.