Chapter 4

Allocation Problems in Parallel Computers
4.1 Overview

- In the early nineties parallel computing was characterized by the following properties:
  - **Machine dependent programming**
    The programmer had to explicitly consider size, type and architecture of the target machine.
  - **Manual allocation**
    The programmer himself was responsible for the mapping of logical objects to physical objects.
  - **Monoprogramming**
    At any point in time only one parallel program could be executed, occupying the entire machine.
- This characterization corresponds to the situation of sequential programming in the sixties.
- System software should make parallel computing as efficient and comfortable as conventional sequential programming.
Allocation Problem

parallel program
parallel program
parallel program

parallel machine
Problem and Problem components

- An allocation problem is described by four components:

1. Machine model \( M \)
2. Load model \( L \)
3. Allocation relation \( R \)
4. Allocation goal \( G \)
4.2 Machine model

- A parallel computer system can be described by a graph, with the processors as the vertices and the direct processor links as the edges:

\[(P, E^p) \text{ with}\]
\[P \quad \text{set of processors as vertices } (|P| = n)\]
\[E^p \quad \text{set of links as edges}\]

Both vertices and edges can have weights:

\[\mu_i: P \rightarrow R \quad \text{vertex weight processor speed } \text{(e.g. MFlops)}\]
\[\gamma_i: E^p \rightarrow R \quad \text{edge weight transmission speed } \text{(e.g. Mbit/sec)}\]
4.3 Load model

• Load can be described at two levels:
  • Program level set of parallel programs
  • Thread level set of interacting threads of a program

• At thread level a parallel program can be represented (analogously to the machine) as a graph:

• \( L = (T, E^T) \) program graph with
  \( T \) set of parallel threads (tasks, threads) as vertices
  \( (|T| = m) \)
  \( E^T \) set of interaction relations as edges

• Vertex and edge weights are also possible:
  \( b_i: T \rightarrow R \) vertex weight length of thread (e.g. #instructions)
  \( a_i: E^T \rightarrow R \) edge weight communication intensity
  (e.g. bits or packets)
Program Graph

- Two types of program graphs
  - task (=thread) interaction graph or
  - task (=thread) precedence graph

Arrows define communication flow

Arrows define precedence relation
Aircraft engineering: Finite-Element-method
Airfoil (Finite-Element Method)
Sieve of Erathostenes (Calculation of primes)
Example TPG

Gaussian Elimination Method (LES)

[Diagram showing a triangular graph with nodes labeled A1, A2, A3, ..., A_n]
Example TPG

Application from Molecular Biology
Program Phase Graph (formal)

- Program phase graph
  \[ PPG := (S, E^S) \]
  \[ S \quad \text{Set of Phases} \]
  \[ E^S \quad \text{Phase transitions} \]
  \[ p_{ij} \quad \text{transition probabilities} \]

- Each phase consists of a TIG:
  \[ s_i := (T_i, E^{T_i}) \quad \forall \ s_i \in S \]

- To make sure that the phases are connected to each other, we request that two adjacent phases have at least one thread in common.
  \[ (s_i, s_j) \in E^S \Rightarrow \exists \ t: t \in T_i \land t \in T_j \]
Parallelism profile

- If the communication behavior is unknown or irrelevant, the program description is reduced to the (dynamic) number of threads.
- If in turn the threads are distinguished from each other, the number of threads (parallelism degree) is sufficient.
- For a dynamic parallelism degree we obtain the parallelism profile (known from chapter 3).

*Parallelism degree \( p(t) \)*

\[ p_{\text{max}} \]

\[ p_{\text{min}} \]
Example: Quicksort on 16 Processors

Average Number of Busy Processors
Example: Fine grain Parallelism

Fig. 7. Parallelism in three consecutive iterations of the VA3D program.
4.4 Allocation

Let be

- \( \text{PCG} = (P, EP) \) \hspace{1cm} \text{The processor connection graph with } \ P \text{ set of processors, } |P| = n \\
- \( A := \{A_1, A_2, \ldots, A_q\} \) \hspace{1cm} \text{the load consisting of a set of parallel programs} \\
- \( T_i \) \hspace{1cm} \text{the set of threads of program } A_i \\

An allocation can take place on the program level or on the thread level.
Program Allocation

- $\varphi: A \rightarrow \wp(P)$ mapping of programs to subsets of processors
- $\varphi(A_i)$ is the processor set allocated to program $A_i$. It is called the **Territory** of $A_i$.
- $\varphi$ is called disjoint, if $\forall i \neq k : \varphi(A_i) \cap \varphi(A_k) = \emptyset$
Program Allocation

• A disjoint program allocation is called **partitioning**. (The processors not allocated by $\phi$ form the so-called **free partition**).

• A territory $\phi(A_i)$ is called **contiguous**, if the subgraph of the PCG defined by the territory is connected.

• A program allocation $\phi$ is called **contiguous**, if $\phi(A_i)$ is contiguous for all $i = 1,.., q$.

Sometimes topological aspects are irrelevant:

A **quantitative partitioning** only decides, **how many** processors each program obtains:

$$\chi : A \rightarrow \{1,\ldots,n\} \text{ with } \sum_{i=1}^{q} \chi(A_i) \leq n$$
Within each program, each thread must be assigned to exactly one processor: \( \pi : T \rightarrow P \)

If \( \pi \) is injective, the allocation is called **injective** (one- (or zero)-to-one), otherwise **contractive** (many-to-one).

threads T

processors P
Thread Allocation

- For a contractive allocation there is often an intermediate step which determines which threads are mapped to the same processor (Contraction, Grouping, Clustering).

![Diagram showing thread allocation from threads T to processors P](image)
Allocation Problem

In multiprogramming operation, an allocation problem can consist of four steps that have to be solved one after the other:

1. Quantitative Partitioning
   • Which program obtains how many processors?

2. Qualitative Partitioning
   • Which program obtains which processors?

3. Clustering (Contraction) within the Programs
   • Which threads are grouped together?

4. Injective Allocation
   • Which thread group is mapped to which processor?
4.5 Goals

List of typical objective functions

- response time RT $\rightarrow$ min
- execution time ET $\rightarrow$ min
- communication cost CC $\rightarrow$ min
- utilization UT $\rightarrow$ max
- Speed-up SU $\rightarrow$ max
- throughput TP $\rightarrow$ max
- load unbalance LU $\rightarrow$ min
- .....  

Since some quantities are contained in others and some are contradictory, it is reasonable to define combinations:

- Arithmetic combination, e.g. weighted sum
- Logical combination using restrictions
  - E.g., ET $\rightarrow$ min $\mid$ LU < 2
4.6 Allocation Algorithms

- An allocation algorithm is described by the problem it is supposed to solve and some additional properties:

- Optimality:
  - An algorithm is called **optimal**, if the optimality of the solution is guaranteed.
  - Otherwise it is called **suboptimal**.
  - Suboptimal algorithms can be divided into two classes:
    - An algorithm is **approximate**, if it finds an optimal solution only approximately. However, an error bound must be provided.
    - If we are neither able to guarantee optimality nor to specify an error bound, the algorithm is called **heuristic**.

- Structure
  - If there is only one instance that has global information and decides about the global allocation then the algorithm is called **central**.
  - **Decentralized** or **distributed** algorithms can be further subdivided into
    - **hierarchical algorithms**
    - **cooperative algorithms** (peer-to-peer)
4.7 Application Areas

Another aspect is the question, at what time the allocation is taking place.

- Offline allocation
  - Optimization problem is formulated explicitly and solved.
- Allocation at compile time
  - Compiler knows the communication and data dependency structure of the parallel program.
- Allocation at start time
  - At this point of time the current load situation is known and can be taken into account.
- Allocation at run-time
  - Data dependent behavior can be collected during program execution (monitoring) resulting in an adaptive dynamic allocation (start new threads, migrate threads).
Further References

• Heiss, H.-U.: Processor Allocation in Parallel Computers (in German) Prozessorzuteilung in Parallelrechnern, Bibliographic Institute, Mannheim, 1994