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:

\[(P, E^P)\]

- \(P\): set of processors as vertices \((|P| = n)\)
- \(E^P\): set of links as edges

Both vertices and edges can have weights:

- \(\mu_i: P \rightarrow R\) vertex weight processor speed (e.g. MFlops)
- \(\gamma_i: E^P \rightarrow R\) edge weight transmission speed (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
Example TIG

Aircraft engineering: Finite-Element-method
Example TIG

Airfoil (Finite-Element Method)
Example TPG

Sieve of Erathosthenes (Calculation of primes)
Example TPG

Gaussian Elimination Method (LES)

Barry Linnert, linnert@inf.fu-berlin.de, Cluster Computing SoSe 2017
Example TPG

Application from Molecular Biology
Program Phase Graph (formal)

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

- 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).

\[ T(\infty) \]

Barry Linnert, linnert@inf.fu-berlin.de, Cluster Computing SoSe 2017
Example: Quicksort on 16 Processors
Example: Fine grain Parallelism

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

Let be

- $\text{PCG} = (\text{P}, \text{EP})$  
  The processor connection graph with $\text{P}$ set of processors, $|\text{P}| = n$
- $A := \{A_1, A_2, \ldots, A_q\}$  
  the load consisting of a set of parallel programs
- $T_i$  
  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 $\varphi$ form the so-called **free partition**).
- A territory $\varphi(A_i)$ is called **contiguous**, if the subgraph of the PCG defined by the territory is connected.
- A program allocation $\varphi$ is called contiguous, if $\varphi(A_i)$ is contiguous for all $i = 1,.., q$.

Sometimes topological aspects are irrelevant:

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

$$\chi : A \rightarrow \{1,\ldots, n\} \text{ with } \sum_{i=1}^{q} \chi(A_i) \leq n$$
Allocation at Thread Level (Mapping)

• 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-to-one), otherwise **contractive** (many-to-one).
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 contractive allocation]

threads T

Contractive allocation

processors P
Allocation Problem

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

- **Quantitative Partitioning:**
  - Which program obtains how many processors?

- **Qualitative Partitioning**
  - Which program obtains which processors?

- **Clustering (Contraction) within the programs**
  - Which threads are grouped together?

- **Injective Allocation**
  - Which thread group is mapped to which processor?
4.5 Goals

List of typical objective functions

- response time RT → min
- execution time ET → min
- communication cost CC → min
- utilization UT → max
- Speed-up SU → max
- throughput TP → max
- load unbalance LU → 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 → min | 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: