Models of parallel computation

lecture 14

Why Model?

"the purpose of modeling is to capture the salient characteristics of phenomena with clarity and the right degree of accuracy to facilitate analysis and prediction"
[Models of Parallel Computation A Survey and Synthesis. BM Maggs, LR Matheson, RE Tarjan]

- A **computational model (abstract machine model)** must define an execution engine powerful enough to produce a solution to the relevant class of problems.

- Problems can come from many different domains e.g. mathematical biological logical
    - an abstract problem specification => the design of a set of steps - an algorithm –
    - translation from problem to computational algorithm requires a model of computation
    - such a model needs to reflect the main computing characteristics of practical computing platforms

- e.g. RAM (Random Access Machine) model ->Von Neumann

A **programming model**

- provide constructs which can elegantly and ~provably translate and preserve algorithmic intentions.
  - a set of rules or relationships that defines the meaning of a set of programming abstractions
- These abstractions are manifested in a **programming language** which is an **instantiation of that programming model**
- A primary objective is to allow reasoning about program **meaning** and **correctness**
  - For example the rules of lambda calculus define the meaning of functions and applications which serve as the basis for the abstractions found in several functional programming languages
  - **Practical programming models** often ***lack such rigor***.
- Examples:
  - the imperative procedural model posits constructs such as *arrays, control structures, procedures, and recursion* => the programming languages such as C are designed within this model
  - the constructs: *Lists Cons Apply* are encompassed by a functional programming model

Model of execution

translation of the high level language constructs into machine dependent executable instructions

- This translation process is usually facilitated not by the use of a programming model but by the use of a **model of the execution** engine
- These machine models are used to **tune the performance** of a programming language and typically **express the cost** of crucial machine operations
- In the realm of parallel computing the demand for performance has begun to realign the focus of language development to include both goals expressability and performance.

Architectural models (or performance models or hardware models)

- describe a class of models used for a broad range of design purposes such as
  - language implementation and
  - machine design
- reflects the detailed execution characteristics of actual or envisioned computers
- covers a wide range from fairly high level representations on an architectural level to instruction execution models or even detailed component models
- primarily used in machine design and the objective of the models is almost exclusively the performance
- in parallel computing is used
  - to compare alternative architectures often for a given class of problems or
  - to predict the performance of a specific machine on a set of algorithmic strategies.
- Performance analysis using these models expresses the design characteristics and concerns of evolving technologies and provides feedback into many aspects of task solution including the design of the machine itself

MPC - Models of Parallel Computation

[Models and Languages for Parallel Computation D. B. SKILLICORN,D.TALIA]

MPC - Model of Parallel Computation

- MPC = is an interface separating high-level properties from low-level ones.
- MPC = a model is an abstract machine providing certain operations to the programming level above and requiring implementations for each of these operations on all of the architectures below.
- It is designed to separate software-development concerns from effective parallel-execution concerns and provides both abstraction and stability.
    - Abstraction arises because the operations that the model provides are higher-level than those of the underlying architectures, simplifying the structure of software and reducing the difficulty of its construction.
    - Stability arises because software construction can assume a standard interface over long time frames, regardless of developments in parallel computer architecture.
- MPC forms a fixed starting point for the implementation effort (transformation system, compiler, and run-time system) directed at each parallel computer.
- MPC insulates those issues that are the concern of software developers from those that are the concern of implementers.
- implementation decisions, and the work they require, are made once for each target rather than once for each program.

Requirements for a MPC

- Easy to Program
- Software Development Methodology
- Architecture-Independent
- Easy to Understand
- Efficiently Implementable
- Cost Measures

Operations types

- Decomposition
- Mapping
- Communication
- Synchronization

Communication variants

- Message passing
- Transfers through shared memory
- Direct remote-memory access

Classification criteria

Levels of abstraction

- Since a model is just an abstract machine, models exist at many different levels of abstraction.

Program Structure

—models in which thread structure is dynamic;

—models in which thread structure is static but communication is not limited;

—models in which thread structure is static and communication is limited.

Categories

1) Models that abstract from parallelism completely
2) Models in which parallelism is made explicit, but decomposition of programs into threads is implicit (and hence so is mapping, communication, and synchronization)
3) Models in which parallelism and decomposition must both be made explicit, but mapping, communication, and synchronization are implicit.
4) Models in which parallelism, decomposition, and mapping are explicit, but communication and synchronization are implicit
5) Models in which parallelism, decomposition, mapping, and communication are explicit, but synchronization is implicit
6) Models in which everything is explicit

1) Models that abstract from parallelism completely

- Such models describe only the purpose of a program and not how it is to achieve this purpose.
- Software developers do not even need to know if the program they build will execute in parallel.
- Such models are necessarily abstract and relatively simple, since programs need be no more complex than sequential ones.

2) Models in which parallelism is made explicit, but decomposition of programs into threads is implicit (and hence so is mapping, communication, and synchronization)

- In such models, software developers are aware that parallelism will be used and must have expressed the potential for it in programs, but they do not know how much parallelism will actually be applied at run-time.
- Such models often require programs to express the maximal parallelism present in the algorithm and then
  - the implementation reduces that degree of parallelism to fit the designated architecture, at the same time
  - working out the implications for
    - mapping,
    - communication, and
    - synchronization.

3) Models in which parallelism and decomposition must both be made explicit, but mapping, communication, and synchronization are implicit.

- Such models require decisions to be made about the breaking up of available work into pieces, but
    - they relieve the software developer of the implications of such decisions.

4) Models in which parallelism, decomposition, and mapping are explicit, but communication and synchronization are implicit.

- Here the software developer must not only
  - break the work up into pieces, but must also
  - consider how best to place the pieces on the designated processor.
- Since locality often has a marked effect on communication performance, this almost inevitably
  - requires an awareness of the designated processor's interconnection network.
- It becomes hard to make such software portable across differffent architectures.

5) Models in which parallelism, decomposition, mapping, and communication are explicit, but synchronization is implicit.

- Here the software developer is making almost all of the implementation decisions, except that
    - fine-scale timing decisions are avoided by having the system deal with synchronization.

6) Models in which everything is explicit

- Here software developers must specify all of the detail of the implementation.
    - it is extremely difficult to build software using such models, because both correctness and performance can only be achieved by attention to vast numbers of details.

Models

- **Nothing Explicit, Parallelism Implicit**
  - Dynamic Structure
    - Higher-order Functional-Haskell
    - Concurrent Rewriting—OBJ, Maude
    - Interleaving—Unity
    - Implicit Logic Languages—PPP, AND/OR,
    - REDUCE/OR, Opera, Palm, concurrent constraint languages
  - Static Structure
    - Algorithmic Skeletons—P3L, Cole, Darlington
  - Static and Communication-Limited Structure
    - Homomorphic Skeletons—Bird-Meertens Formalism
    - Cellular Processing Languages—Cellang,
    - Carpet, CDL, Ceprol
    - Crystal

- **Parallelism Explicit, Decomposition Implicit**
  - Dynamic Structure
    - Dataflow—Sisal, Id
    - Explicit Logic Languages—Concurrent Prolog, PARLOG, GHC, Delta-Prolog,
    - Strand
    - Multilisp
  - Static Structure
    - Data Parallelism Using Loops—Fortran variants, Modula 3*
    - Data Parallelism on Types—pSETL, parallel sets, match and move, Gamma, PEI, APL, MOA, Nial and AT
  - Static and Communication-Limited Structure
    - Data-Specific Skeletons—scan, multiprefix, paralations, dataparallel C, NESL, CamlFlight
- **Decomposition Explicit, Mapping Implicit**
  - Dynamic Structure
  - Static Structure
    - PRAM, BSP, LogP
  - Static and Communication-Limited Structure.

Models

- **Mapping Explicit, Communication Implicit**
  - Dynamic Structure
    - Coordination Languages—Linda, SDL Non-message Communication Languages—
      - ALMS, PCN, Compositional C11
    - Virtual Shared Memory
    - Annotated Functional Languages—Paralf
    - RPC—DP, Cedar, Concurrent CLU, DP
  - Static Structure
    - Graphical Languages—Enterprise, Parsec, Code
    - Contextual Coordination Languages—Ease, ISETL-Linda, Opus
  - Static and Communication-Limited Structure
    - Communication Skeletons

- **Communication Explicit, Synchronization Implicit**
  - Dynamic Structure
    - Process Networks—Actors, Concurrent
    - Aggregates, ActorSpace, Darwin
    - External OO—ABCL/1, ABCL/R, POOL-T,
    - EPL, Emerald, Concurrent Smalltalk
    - Objects and Processes—Argus, Presto, Nexus
    - Active Messages—Movie
  - Static Structure
    - Process Networks—static dataflow
    - Internal OO—Mentat
  - Static and Communication-Limited Structure
    - Systolic Arrays—Alpha

- **Everything Explicit**
  - Dynamic Structure
    - Message Passing—PVM, MPI
    - Shared Memory—FORK, Java, thread packages
    - Rendezvous—Ada, SR, Concurrent C
  - Static Structure
    - Occam

Analysis

- Work on low-level models, in which the description of computations is completely explicit, has diminished significantly.
- There is a concentration on models in the middle range of abstraction.
    - there are tradeoffs among expressiveness, software development complexity, and run-time efficiency.
    - Presumably a blend of theoretical analysis and practical experimentation is the most likely road to success
- There are some very abstract models that also provide predictable and useful performance on a range of parallel architectures.
    - Their existence raises the hope that models satisfying all of the required properties can eventually be constructed.