


TYPE ARCHITECTURES, SHARED MEMORY AND THE COROLLARY OF MODEST POTENTIAL

Lawrence Snyder (1986)
Dept. of CS, University of Washington
Presented by Joe Mayo

THE FUNDAMENTAL LAW OF PARALLEL COMPUTATION

- A parallel solution utilizing p processors can improve the best sequential solution by at most a factor of p .
- Otherwise known as Amdahl's Law
- Implies that superlinear speedup implies the existence of a faster sequential solution.




THE COROLLARY OF MODEST POTENTIAL


- Assume time is a constant
- Assume best possible speedup; no overhead
- If a problem takes $t = cn^x$ sequential steps
- And an increase of p processors allows an increase in problem size by a factor of m :

$$t = c(mn)^x/p$$

Then the problem size can increase by a factor of:

$$m = p^{1/x}$$


THE COROLLARY OF MODEST POTENTIAL

- Example:
 - Sequential problem:
 - $t = cn^4$ (e.g. 3D simulation over time => 4 dimensions)
 - To increase the problem size by two orders of magnitude requires, optimistically, 10^9 processors!
- 

THE COROLLARY OF MODEST POTENTIAL

In the context of superlinear problems, parallel computation offers only a modest potential benefit!

- In the context of problems running in a higher degree or is exponential, the potential improvement is correspondingly more modest.
- The point is not that parallel computing is futile, but that it's very important to avoid overhead both in languages and architectures so the programmer can utilize the most efficient facilities.




PARALLEL VS. SEQUENTIAL ALGORITHMS


- There's an inherent difference between parallel and sequential algorithms:
 - *Sequential*: Exhibit artificial ordering restrictions
 - This is due to assumptions of how instructions are ordered
 - *Parallel*: Exhibit the weakest possible ordering constraints



PARALLEL LANGUAGE EXTENSIONS

- Often not sufficient due to constraining the parallel constructs with the sequential semantics
 - A parallel language might reduce to a sequential language when run on a single process
 - But you can't guess a parallel generalization beginning with a sequential language
 - Best to start with a new parallel language than trying to add extensions to a sequential language.
- 

PARALLEL LANGUAGES

- Goals:
 1. Conceptually clean abstract machine model that matches coherent subclass of parallel algorithms
 2. Efficiency of a program's implementation on a particular machine
 - These goals produce a gap between the abstraction (i.e. overhead) and offering efficient lower level facilities to improve performance.
- 

ENTER TYPE ARCHITECTURES

- What's helpful is to have a machine model to build upon.
 - Like the Von Neumann machine is for sequential machines
- What's needed is a Type Architecture:
 - An idealized parallel machine
 - Standardization of the hardware/language interface




TYPE ARCHITECTURE REQUIREMENTS

- Must accurately reflect costs
 - Using unpractical costs will lead programmers to making poor choices in regards to algorithm choice
 - When you can't rely on costs, it breaks the effectiveness of the type architecture
- Must display the principal structural features of the architectures
 - Features hidden by the type architecture are hidden from the language
 - Without these facilities, the programmer can't tell the compiler how the program should be run
 - Thus compiler must employ only generalized, inefficient translations




CANDIDATE TYPE ARCHITECTURE (CTA)

A finite set of sequential computers connected in a fixed, bounded degree graph, with a global controller

- *Finite set:* Programmers assume no bound (like we do with memory), compiler handles the case when it is exceeded
 - *Sequential Computers:* von Neumann type architecture is presumed. “implements a sequential instruction stream”
 - *Connected:* Ability to send a simple value between computers efficiently and simultaneously
- 

CANDIDATE TYPE ARCHITECTURE (CTA)

A finite set of sequential computers connected in a fixed, bounded degree graph, with a global controller

- *Fixed, bounded degree graph:* excludes busses. Bounded by the finite set of machines.
 - *Global controller:* provides a weak control over computers consistent with multiple independent programs and asynchronous control
- 

CANDIDATE TYPE ARCHITECTURE (CTA)

- Accurately reflects costs
 - Inherits from Von Neumann type architecture
 - Defines unit-cost memory access to a local memory
 - Mute on the existence of global (i.e. shared) memory
- Exposes underlying structure
 - Specifies structure as a graph with a global controller
 - Capable of fine-grained communication



DESIGNING A LANGUAGE ON THE CTA

- Extending Existing Languages:
 - References to nonlocal data values must be converted to sends and receives by the compiler
 - Synchronization and other control operators must be implemented; hard to do
 - CTA doesn't favor centralized facilities as semaphores
- New Languages:
 - Inclined to recommend definition of a new language
 - The language can shift burden of allocation, synchronization, etc onto the programmer



SHARED MEMORY (SM) IN THE CTA

- No shared memory in CTA, but not excluded
- Language designers can/should provide SM as a language abstraction defined in terms of the CTA
- Architects can help by providing automatic routing hardware to support nonlocal references
- Take away: CTA shouldn't prevent the programmer from being able to use a SM model



THE PROGRAMMER:

- Accommodating the programmer's desire for convenience in a CTA-type language depends on the effective use of known facilities and the development of new, more powerful programming abstractions
- CTA Serves the need for efficient parallel programming mandated by the Corollary of Modest Potential



ABSENCE OF ANY SPECIFIC CHOICE OF GRAPH

- Each family of graphs have their own assets and liabilities
- No graph family has proven itself to be the best
- Simplicity often translates to a favorable cost-benefit
- Benefit for architects: No constraints on design decisions
- Can be a benefit to model program as a graph



CONTRACTION & MAPPING

Contraction: Task of mapping a member of a graph family down to a smaller member of the family

- + Good contractions for popular graph families exist
- + There exists automatic methods of contraction
- + Straightforward for programmers to incorporate contractions into their programs
- Program and architecture graphs must be from same family
 - Or accurate costs and visible structure may not be realizable



CONTRACTION & MAPPING

Mapping: Task of embedding a graph G (guest) into a graph H (host) so that the vertices map 1-to-1 and the edges of G map to paths of H



SUMMARY

- Choice of language affects the algorithms chosen
- Prefer new parallel languages over extending sequential languages
- Choosing a Type Architecture is key:
 - Must accurately reflect costs
 - Expose the machine structure => performance
- CTA meets these requirements
- Shared memory can/should be modeled via CTA
- Corollary of Modest Potential
 - Leads to intense interest in avoiding inefficiency

