Programming Models for Future High Performance Computing Systems

John Gurd
University of Manchester
United Kingdom
Darlington’s Dream (c.1985)

functional programming languages

plus

program transformation
Key Requirement

To cope with continuing divergence of trends in:

• applications (what we want to compute)
• attitudes (how we expect, or are expected, to work)
• systems (what is provided to help us)
Application Trends

• large-scale
  – much data
  – many operations

• mixed models
  – multiple physics
  – numerical + symbolic data/info/knowledge

• better science
  => more computing => performance
Attitude Trends

• maintainability
  => abstraction away from architecture

• reusability
  – especially of design

• predictability
  – error/accuracy
  – solution time
System Trends

- **distribution**
  - more than parallelism
- **heterogeneity**
  - more than homogeneity
- **dynamic multi-computing**
  - more than batch
- **scale**
  - more
Tool Trends

• in the midst of this divergence, clearly tools play an important role, however . . .

• tools are always playing “catch-up”
What’s Wrong?

• attention still on performance (not even on expressing the application well, let alone on maintainability and design reuse)
• application-oriented approaches currently sacrifice performance for other things
• performance-oriented approaches currently rely on explicit control of architectural features
• performance-oriented tools lag behind architecture and application innovations
Analysis

• architecture trends are all in directions that make explicit programming more difficult, and tools cannot (be expected to) keep up;
• hence it is becoming ever more essential to develop programs that are independent of (abstracted away from) hardware, and
• to find ways to automate generation of high performance implementations from these
Aspects (a way forward?)

“Aspect Oriented Programming (AOP) is based on the idea that computer systems are better programmed by separately specifying the various concerns (properties or areas of interest) of a system and some description of their relationships, and then relying on mechanisms in the underlying AOP environment to weave or compose them together into a coherent program.”

[CACM special issue, Oct.2001]
Gurd’s Dream

aspect (inc. performance) oriented codes

plus

automated, dynamic code “weaver”
Legacy Code?

given that legacy high performance software is bespoke to old architecture, surely the main question is whether we should even try to efficiently port it to new architectures – if we ever realised Gurd’s Dream, we would re-write at a higher level and let the weaver do the hard work in the interim, the usual pragmatic methods will have to be used: define interfaces; wrap up in the form of components; compose into new programs