Dynamic Accommodation of Performance, Power, and Reliability Tradeoffs

Special ACK to:
Vivek Sarkar (co-PI)
& team, Rice
Kath Knobe, Intel

U.S. Dept. of Defense

Rob Knauerhase
Intel Labs

Fifth Annual Concurrent Collections Workshop
September 24, 2013

Overview

- Bit of Motivation
- Bit of History
- The Big Idea
- Challenges (practical and practical)
- Integration with coordinaton language, programming language, compiler

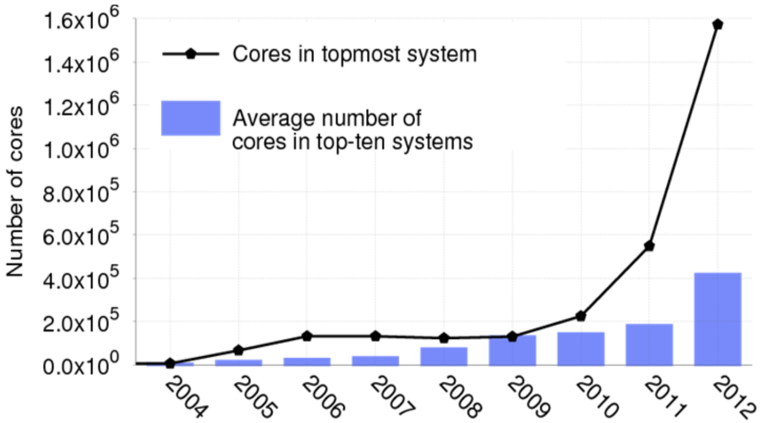
Notes / Disclaimers:

1) I am not really a CnC programmer, but I play one on TV.
2) Much of this is work in progress or just starting
3) We overlap with and like, but are distinct from other efforts

Motivation

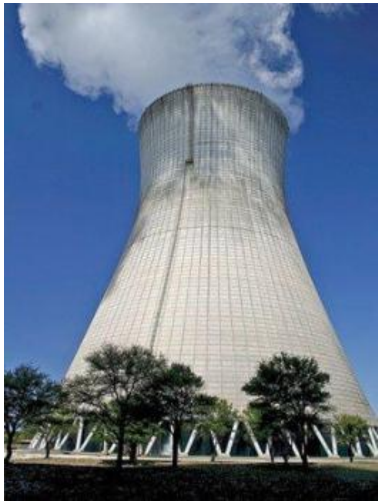
- State of the world
  - moving to extreme scales (exa-*)
  - high-core-count machines
    - (HPC, server, and even mobile!)
  - use of increased concurrency with lower clock frequency
  - "dark Si" – performance and functional heterogeneity; NTV and AxC
  - computation becomes relatively cheap ("free")
  - data movement is a major contributor to energy usage
    - therefore, locality important for both performance and energy efficiency

Top500® List, November 2012



Source: http://researcher.watson.ibm.com/researcher/view_project_subpage.php?id=3669

Motivation



- New challenges
  - power becomes major limiting factor
    - extrapolating current trends to 2020's, power required for exa-scale performance

      ⇒ 530 Megawatts(!)
  - reliability also more challenging
    - current HPC systems spend 28% of time checkpointing
      - (can) introduce a giant system-wide barrier
      - often affect design/implementation of entire algorithm
- And don't forget about performance ☺

Motivation

- Now, add in changing system-wide policies
  - minimize peak power (e.g. for infrastructure or cost reasons)
  - minimize energy consumed (e.g. $$$)
  - precise or “good enough” results
  - changes in nature of mission
- And dynamically-changing environments
  - diesel generator in the field & mission
  - datacenter operating $ vs. urgency
  - image processing (diagnosis vs. pre-surgery) *(thanks, Alina, for the example!)*

Hmmmmmmmm.
Could a new programming model help with this?

Motivation

- Results: unheard-of complexity
  - imagine a *million* threads
  - then try to optimize them
  - *then* consider different platforms
- “Hero” programmers can accommodate this sort of thing
  - but heroes are *rare* and therefore *expensive*
  - and worse, they don’t scale
    - across algorithms
    - across platforms
    - across policies

Motivation

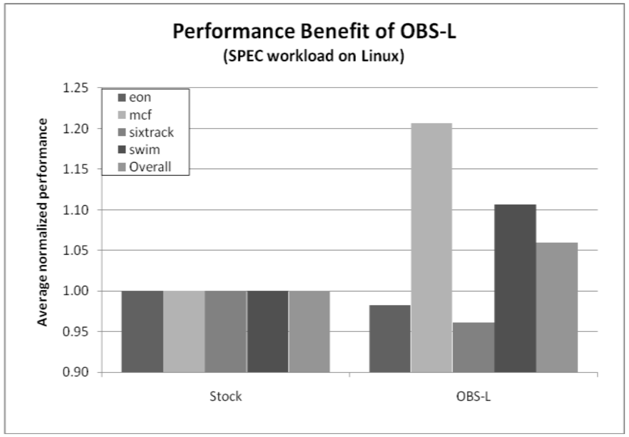
There must be a better way, amirite?!?

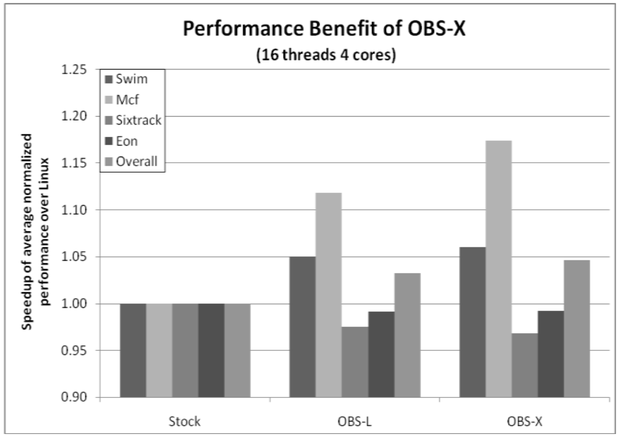
- A new paradigm?
  - one that will relieve some of the programmer burden
    - (esp. since HPC folk are interested)
  - one that will lessen the need for heroic skill/expertise
  - one that lends itself to adaptation to dynamic changes
- ...how about a fine-grained, event-driven, adaptive model
  - like maybe OCR? ☺
- ...and how to feed such a beast?
  - maybe CnC? ☺

Background

- OS-level techniques for performance/power improvement
  - monitor usage and contention (e.g. of caches)
  - alter scheduling policy (which queue & position within)

- Bingo! ~6% performance savings on SPEC-style workloads

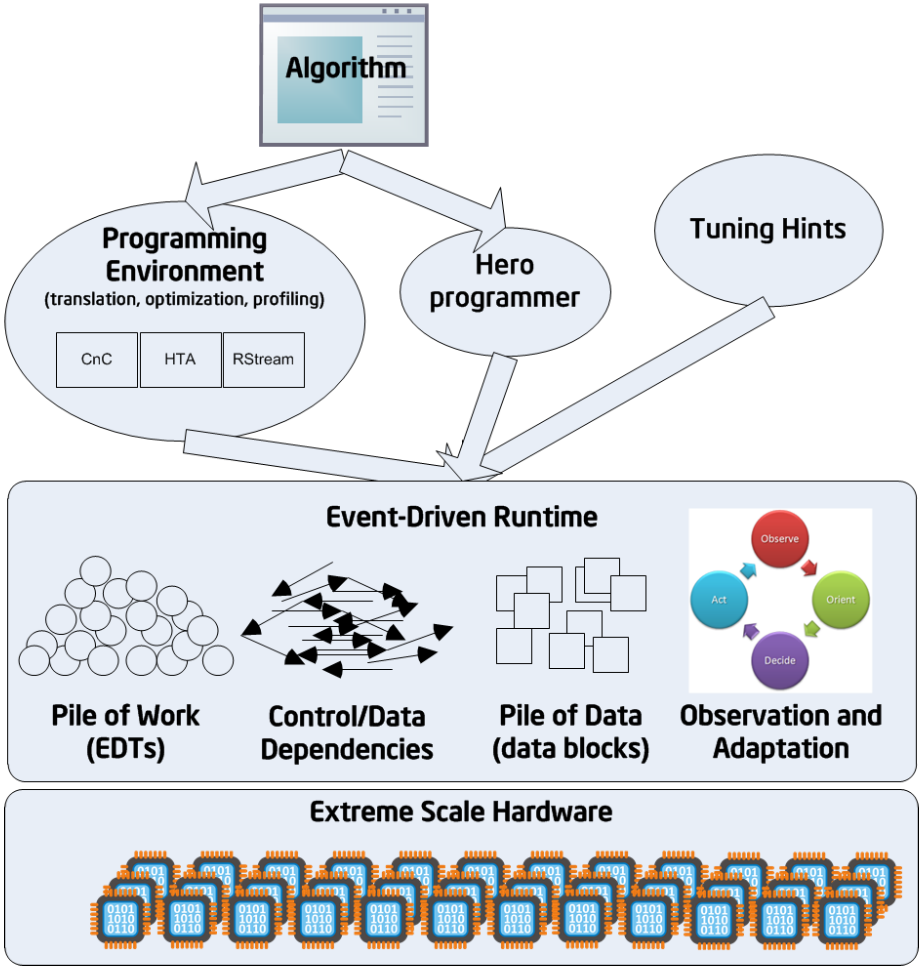




See IEEE Micro, Vol 8 Iss 23, 5/08, *inter alios*

Background

- Dynamic runtime prior work
  - SW research from DARPA/UHPC Runnemede program, later evolved into OCR
  - like magic 8-ball says, "signs point to yes"
- Fine-grained
  - allows many points to "intercept"
- Event driven
  - allows easy accommodation of dynamic dependence



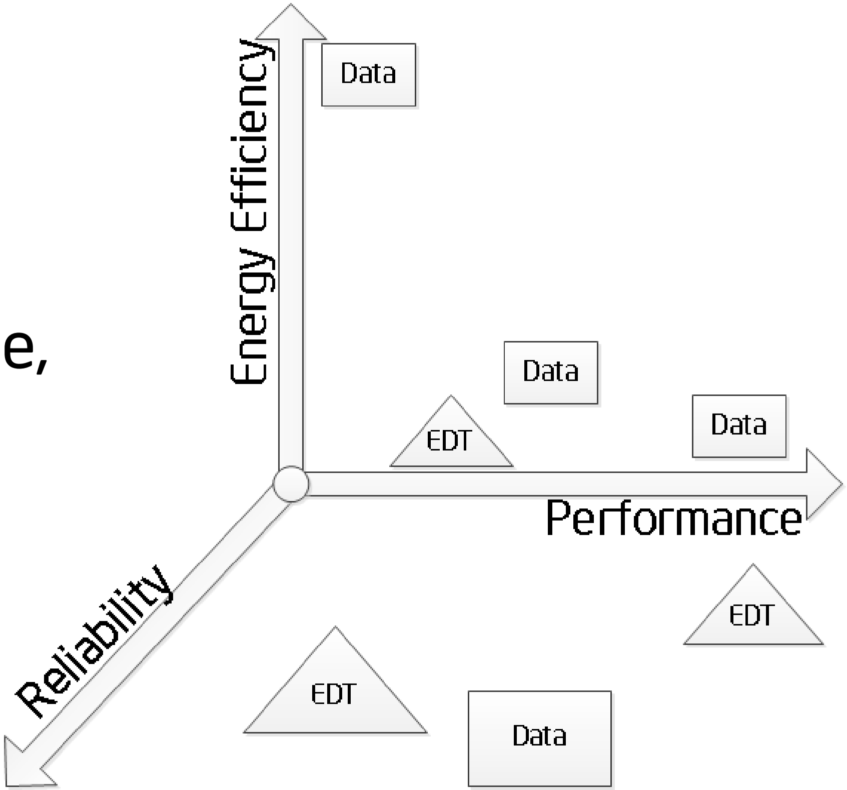
Ongoing/ future

Runtime optimizes based on tuning hints & inferences from events

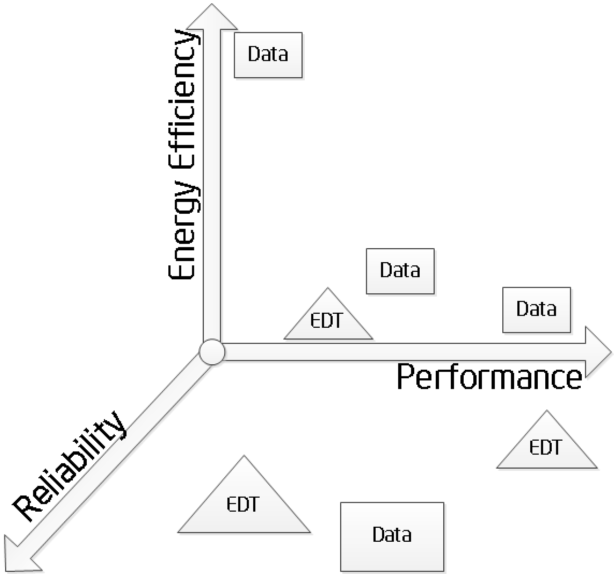
See SC12 BoF, upcoming SC13 BoF, *inter alios*

The Big Idea

- Add features to runtime environment that allow specification of importance of *power, performance, and reliability*
  - imagine a 3-space
    - "good, fast, cheap – pick any 2"
    - not always antagonistic, but even if so, not always linearly
- "Things above" specify importance, per component (task/data)
  - either explicitly by developer
  - or by tuning expert
  - or by intelligent compiler
  - or … … … ?



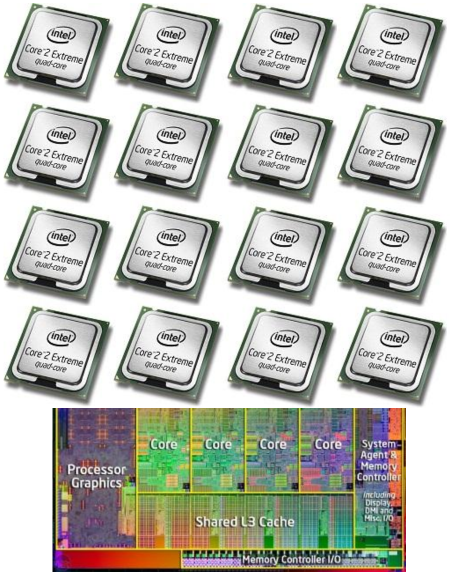
The Big Idea: research hypothesis



Code and data expressed in FGED model

ACR Runtime Environment

Code/data scheduled/mapped to specific resources



Runtime executes components per hints and observations

(Process repeats)

Analytical solutions probably computationally intractible

The Big Idea: questions

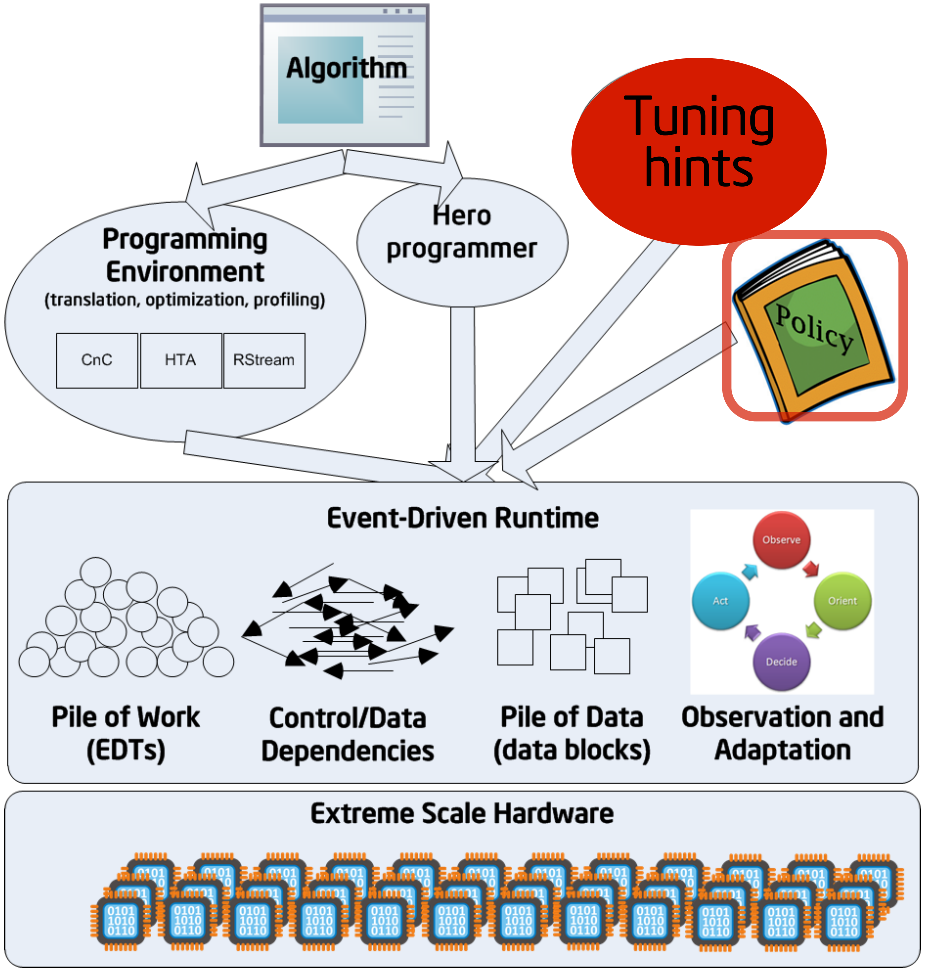
- How to arrange tasks/data within system to minimize latencies or energy costs?
  - start: programmer hints (if she’s smart)
  - refinement: move data to code
  - refinement: move code to data
  - refinement: re-map based on observed frequency of access
  - refinement: choose among *equivalent versions* of an EDT

- How to accommodate *interference* in shared resources
  - including memory busses, caches, scratchpad memories, …

Relies heavily on current PMUs (performance monitoring units), plus roadmap plans *and* input into the roadmap

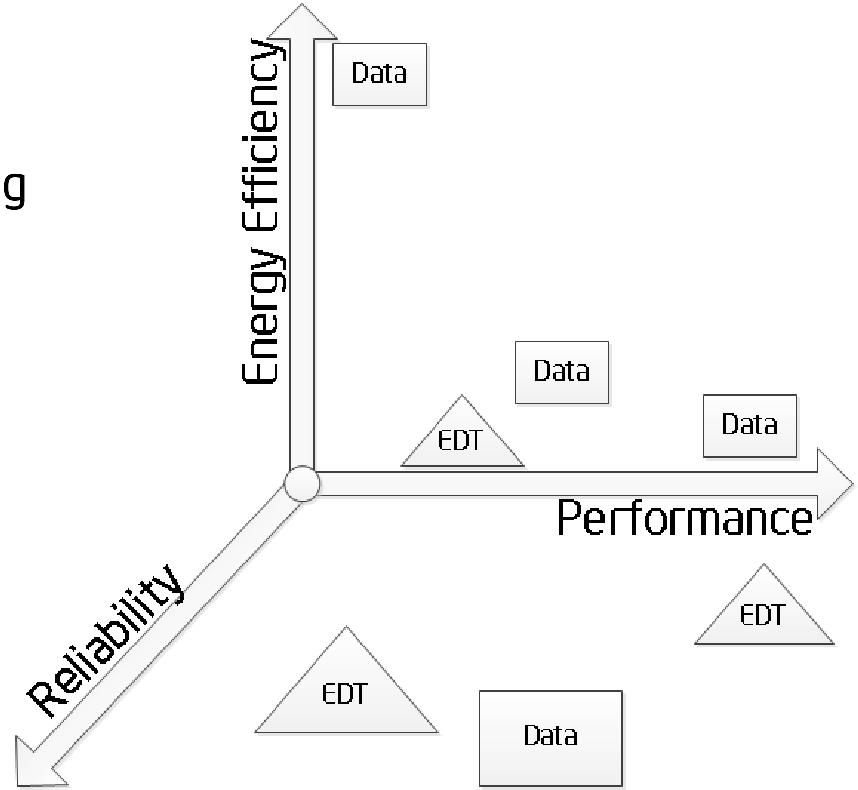
The Big Idea: input parameters

- Build upon existing fine-grained, event-driven system
- System Policy
  - allows expression of attributes described before
- Tuning hints
  - Ongoing focus on placement and affinities
  - Add description for *relative importance* of power, performance, and reliability



The Big Idea: empirical hypothesis

- Runtime system
  - looks first at importance (from position)
    - ex. "job tracker needs to be reliable, leaf computations should be fast"
  - then builds clusters for grouping
    - reduces overhead for later scheduling
  - initially maps components onto particular hardware
    - based on hw availability and nature of sw-component clusters
  - monitors system state, hw usage, & environment
    - move components as needed to heuristically meet spec



Practical Challenges

- Power optimization factors
  - code/data placement for energy (differences from performance?)
    - including what can be "turned off" with different scheduling
  - whether/when to copy data for locality (energy tradeoffs)?
  - possible upcoming hardware features (e.g. per-core or per-FUB gating)
- Reliability factors
  - how does NTV affect reliability?
  - what hardware will help with fine-grained fault detection?
  - what about sw hints for checkpointing per execution frontier
- Combined factors
  - overhead of monitoring, determining adaptation, & *doing* adaptations
  - approximate computing (hw or sw)?

Other Challenges

- application support
  - still a lack of nontrivial app implementations for OCR
- compiler support
  - are there things we can be told from compile time
    - e.g. hints about tiling, loop unrolling, loop perforation, …
- tuning support
  - contemporaneous development of tuning language (both expressibility and syntax)
- implementation overhead
  - do we save more than we consume to determine closer-to-optimal solution?

Discussion (hopefully there's time ?)

- Do we understand what information CnC can provide?
  - are there changes/extensions that would make this easier
- How do we interface our tuning with potential guidance to low-level compiler guidance? Can compiler convey things to us?
- Are there "CnC-compatible" autotuners that would assist or defeat our system?
- Other topics as they arise.

intel®