

For extreme parallelism, your OS is sooooo last-millennium

Rob Knauerhase, *Romain Cledat*, Justin Teller

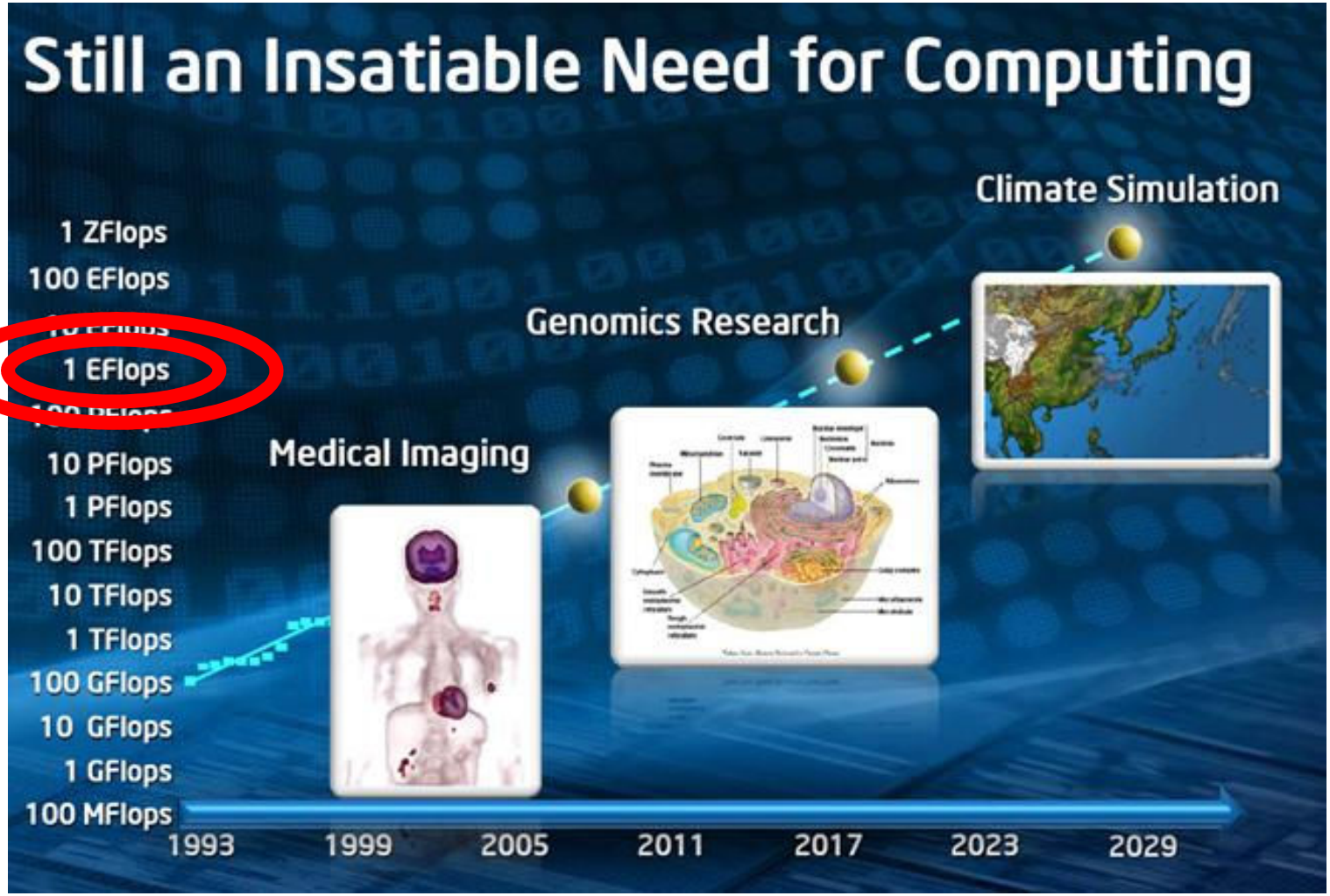
Government Purpose Rights
Purchase Order Number: N/A
Agreement No.: HR001-10-3-0007
Contractor Name: Intel Corporation
Contractor Address: 2111 NE 25th Ave M/S JF2-60, Hillsboro, OR 97124
Expiration Date: None
The Government's rights to use, modify, reproduce, release, perform, display, or disclose this technical data are restricted by paragraphs B (1),(3) and (6) of Article VIII as incorporated within the above purchase order and Agreement. No restrictions apply after the expiration data shown above. Any reproduction of the software or portions thereof marked with this legend must also reproduce the markings. The following entities, their respective successors and assigns, shall possess the right to exercise said property rights, as if they were the Government, on behalf of the Government: Intel Corporation (www.intel.com); ET International (www.etinternational.com); Reservoir Labs (www.reservoir.com); University of California San Diego (www.ucsd.edu); University of Delaware (www.udel.edu); University of Illinois at Urbana-Champaign (www.uiuc.edu).

Outline

- Backdrop:
  - On the road to Exascale
  - Runnemede: the UHPC project at Intel
- Design decision points:
  - Separation of concerns
  - Memory management
  - Threading model
- Where do we go from here?

Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

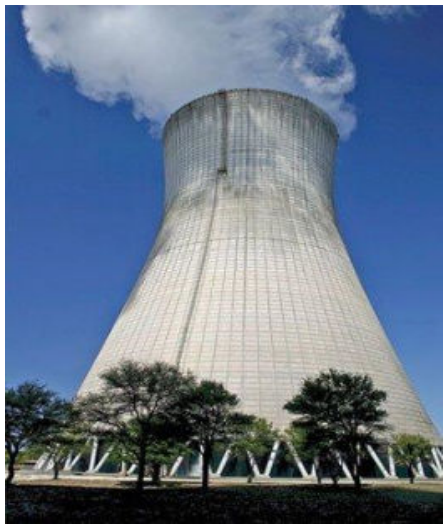
The road to exascale…



Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

… will require a nuclear plant

- Extrapolating current power trends to 2018: 530 MW for an exascale machine
- Is a *nuclear power plant* per system really sustainable?



Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

Principles for current OSes…

- Design philosophies of OSes:
  - Abstract away the hardware
  - Provide common programming API
  - Ensure fairness, isolation, etc.
- Extremely successful for current development:
  - Write once for multiple processor generations
  - Focus on application, *not* nitty-gritty boiler-plate code (memory management, threading, etc.)

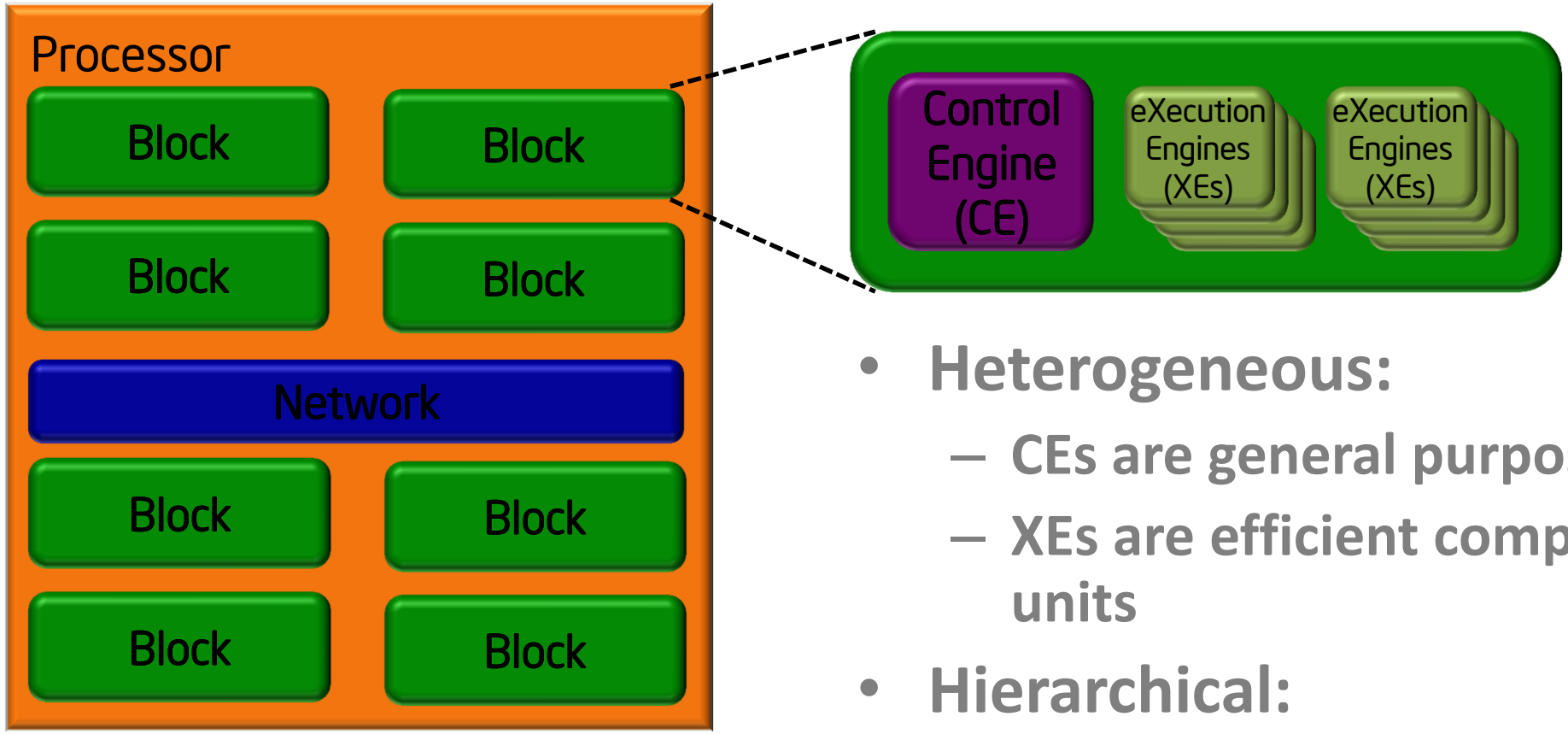
Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

… not adapted for exascale

- Exascale *requires* visibility for energy reasons:
  - into the memory hierarchy
  - into task, not just thread, scheduling
- Exascale does not have the same concerns:
  - compute resources are plentiful (so why share)
  - communication is expensive (not even seen in OS)

Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

The Runnemede project: hardware



- Heterogeneous:
  - CEs are general purpose
  - XEs are efficient computation units
- Hierarchical:
  - Both compute and memory

Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

The Runnemede project: software

- Programming model based on:
  - Small chunks of code (codelets)
  - Dataflow-like dependencies between codelets
- Codelets run uninterrupted on the XEs:
  - Codelets “fire” when their dependencies are met
  - Codelets “satisfy” the dependencies of other codelets

Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

Current hypotheses

- Exascale system software will require:
  - Fine-grained event-driven execution model
  - Sophisticated observation
    - Execution, environment, policies, resiliency, …
  - Dynamic adaptation techniques

Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

Outline

- Backdrop:
  - On the road to Exascale
  - Runnemede: the UHPC project at Intel
- Design decision points:
  - Separation of concerns
  - Memory management
  - Threading model
- Where do we go from here?

Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

Traditional separation of concerns

- Traditionally, resources are time shared between:
  - A kernel that interacts with the hardware
  - User code
- Downsides:
  - Expensive context switch on the same core (kernel vs. user mode)
  - Does not take into account differences in processing requirements for kernel and application

Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

Separation of concerns: proposal

- Resources are plentiful:
  - Use some for “kernel” and others for “user” code
    - CEs run “kernel” code and XEs run “user” code
  - *Spatial* sharing instead of temporal sharing
- Specialize resources:
  - E.g., CEs specialized for queue processing
  - XEs specialized for energy efficiency + custom functionality as needed

Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

Traditional memory management

- Traditionally:
  - Virtual memory is used to abstract actual memory hardware (available amount of RAM, layout, etc.)
  - Heap allocation is done in an architecture agnostic manner (except for some NUMA effort)
  - Storage hierarchies do not adapt to the application
- Loss of *visibility* on the granularity and locality of the memory

Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

Why is visibility important?

- Granularity:
  - Optimize for application's access patterns (not all data fits exactly in a multiple of the page size)
- Location:
  - Give software an explicit view of communication and access costs
  - Allow a programmer (or compiler) to precisely place allocations within a memory hierarchy

Memory management: proposal

- Make data a first-class object:
  - A runtime system can track usage at a programmer/compiler controllable granularity which makes sense from the application's point-of-view (data objects)
  - Data objects can be relocated, garbage collected in a way to minimize energy usage
- Memory management will collaborate with task scheduling

Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

Threading abstraction: constraints

- Avoid context switching
  - Energy inefficient and attempts to “save” a plentiful computing resource
- Avoid hard-binding to hardware resources
  - The system state, workload, and overall power/performance tradeoffs will change during an application lifetime
- Co-location of a thread and its data is key to reducing energy
  - Moving a data element may be more expensive than computing it (for eg. with NTV computation)

Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

Threading abstraction: proposal

- Build a fine-grained, asynchronous task parallelism abstraction at the lowest level:
  - Stop tying parallel tasks to resources
  - Make explicit affinity relationships
  - Make explicit dependency relationships

Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

Threading abstraction: proposal

- Enables the programmer to precisely schedule tasks
  - Expose affinity interface among tasks and data instead of to particular hardware/threads
  - Allow the runtime to adapt to the time-varying capabilities of the underlying hardware
- Explicit dependency interface
  - Similar to TBB task graph and Habanero DDFs
  - Can interface with the memory management portion of the runtime, informing data-movement and task scheduling

Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

Summary

- Current OS design goals are misaligned with Exascale goals
- Replace OS with a lightweight runtime:
  - Strips out unnecessary OS features
  - Focuses on Exascale goals (for example: energy efficiency and resiliency)
  - Provides low-level interfaces to *expose* certain key architectural aspects (relevant for energy)

Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

Outline

- Backdrop:
  - On the road to Exascale
  - Runnemede: the UHPC project at Intel
- Design decision points:
  - Separation of concerns
  - Memory management
  - Threading model
- **Where do we go from here?**

Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

More compiled visibility

- Non-code information is lost in the final binary
- Useful information for Exascale:
  - Different trade-off implementations
  - Capabilities/requirements of various codelets
- Preserving user annotations and compiler deduced information would be helpful at runtime

Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

Non-traditional core use

- Resources are plentiful in Exascale:
  - Explore other uses for cores:
    - Use some to explore computation space and do JIT tuning
    - Dedicate some to smarter scheduling
    - …
  - Explore correct mix of cores

Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

Advanced observation

- Use PMUs to inform scheduling decisions
  - Determine “typical” codelet access patterns
  - Quick fixups if needed for existing scheduling decisions
  - Accommodate dynamic system state
    - Not just load, but also temperature, resiliency, …
  - Accommodate heterogeneity and data-parallelism (codelet version selection)

Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

Enhanced data/task scheduling

- Key goal: reduce communication
- Heuristics to optimally schedule code and place data:
  - Programmer aided
  - Observation/learning based
- Other goals:
  - Distribute heat
  - Discover and recover from errors

Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**

Thank you.

Questions?

Use or disclosure of data contained on this slide is subject to the restrictions on the title page of this presentation. **This research was, in part, funded by the U.S. Govt. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Govt.**