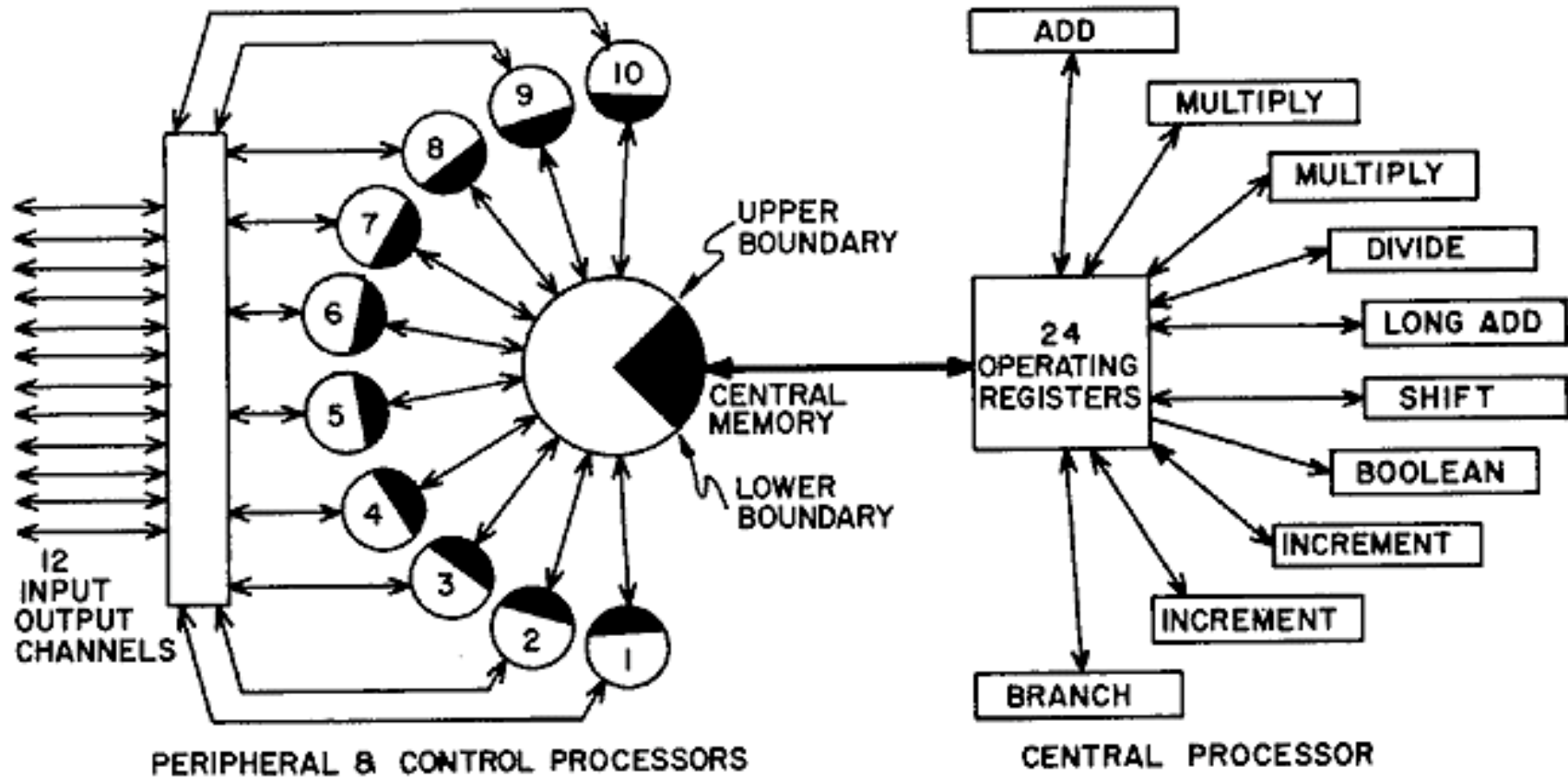


**“You want me to get results on
WHAT?”**

Paul Henning
CCS-2/Roadrunner Project
Los Alamos National Laboratory

Heterogeneous processors are not new.



Abstractions were introduced to reduce complexity.

- **Assemblers allowed mnemonics for machine instructions.**
- **Compilers added translations from HLL to assembly for a single CPU.**
- **Then what happened to hardware?**
 - CISC/RISC.
 - SIMD, superscalar, branch prediction, etc.
 - Deep memory hierarchies.
 - Multiple threads (OS and hardware).
 - SMPs.
 - MPP Clusters.
 - Clusters of hierarchies of heterogeneous processors.

Programming *abstraction* didn't keep up.

- **Programming techniques:**
 - Libraries: share common code.
 - Computational frameworks: DSELs.
- **Tools:**
 - Language/Compilers with parallel support (auto-parallelization, OpenMP, PGAS, etc.)
 - Communication frameworks/libraries.
 - Tendency to be “bottom-up,” following contemporary architecture.
- **All excellent things, but...**

Heterogeneous exacerbates long-standing problems in application abstraction.

- **Focusing on instructions rather than data motion.**
 - Instruction rate hasn't been the bottleneck in ages.
- **Trying to shoehorn system-level issues into compilers and/or libraries.**
 - Compilers translate from HLL to a single ISA.
 - Compilers understand one CPU at a time. All parallelism is due to “glue.”
 - They are very good at optimizing *instructions*.
- **Intermixing the “how” with the “what.” Code becomes brittle when...**
 - ... a communications model/implementation is selected.
 - ... memory is allocated or data structures are embedded in memory.
 - ... a parallel decomposition is selected.
 - ... assumptions are made about processor capabilities (e.g. SIMD).

A proposal for higher-level tools for HPC.

- **Construct tools “over” the compilers: apps that generate apps.**
- **Generate HPC application from:**
 - Domain specific language.
 - Description of a HPC system (implicit or explicit).
 - System “architect” tool does analysis of dataflow, types, *etc.*
 - System “embedder” tool creates imperative HLL+libraries+frameworks+build system+job launch for particular system.
 - Build/execute as usual.
- **Additional benefits:**
 - Do unit analysis, advanced numerical analysis, *etc.*
 - Could generate an optimized simulation for a particular set of inputs.
 - Separate skill sets... let people do what they are good at.
 - Preserves the expert methods knowledge embodied in the calculation.

You really don't have to program everything.

- **Examples:**

- Spreadsheets
- SQL

- **Features:**

- No explicit memory management.
- No embedding of execution environment.
- Don't need low-level access to the execution platform.

Success lies in the limits.

- **Should be possible for limited domains, very challenging in general.**
 - Very different than auto-parallelizing compilers, since we know a lot about the application being produced. Rewriting applications should be fun!
 - Have computers do the boring, redundant bookkeeping.
- **Need to separate the “what” (the calculation) from the “how” (the implementation details):**
 - Functional(-ish) languages are easier to analyze.
 - Data structures are for the convenience of the user, not dictates to the compiler.
 - No explicit memory management by the user.
- **Build on all the other tools/frameworks/methodologies we have been talking about!**
 - Auto-parallelization, PGAS, MPI, concurrency frameworks, simulation frameworks, threading, auto-tuning, *etc.*

Domain-specific application abstraction solves many problems.

- Separating the “what” from the “how” creates portability by delaying the decisions that make code brittle.
- Captures and preserves the “essence” of the calculation being performed.
- Provides an organizational framework that allows experts from different fields to contribute equally.