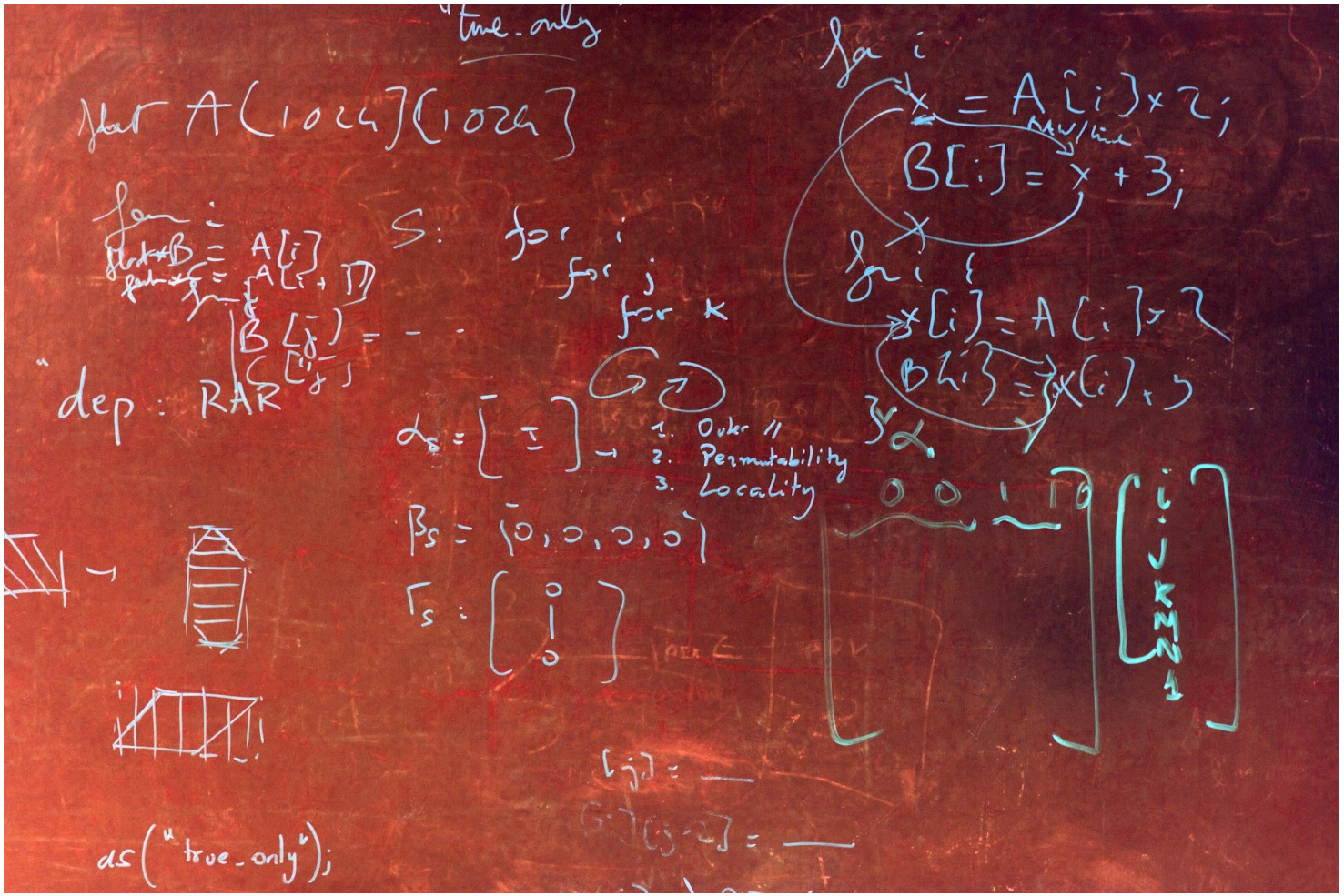
Retrospective on Debugger Support for Bare Metal HPC

George Whiteside and Benjamin Huang



New hardware means new toolchains

- Diverse architectures with diverse programming models
- GPGPU and massive parallelism is common
- **Debuggers provide enormous value to developers**

Architecture overview

- HPC
- No operating system (bare metal)
- High number of threads (10,000s)
- Lots of different kinds of memory

Why did we choose LLDB?

- Part of the LLVM umbrella project
- Full-featured command-line debugger
- Modern, retargetable code base
- Python scripting

Problems

- POSIX environment is assumed, which restricts our use case
- We'll look at these 3 examples
  - Stepping
  - Memory
  - Threading

Stepping

Problem

- No `ptrace` calls to manipulate process being debugged
- Hardware support for stepping is not consistent across architectures

Stepping

Solution

- Disable certain run modes and change default
  - One architecture works best running only one thread at a time (`eOnlyThisThread`)
  - Another architecture can only run all threads (`eAllThreads`)
- Some `ThreadPlans` assume all run modes would be available

Memory

Problem

- LLDB expects virtual memory from OS
  - Homogeneous address space
- Our architectures have more complex, heterogeneous memory
  - **Thread-local memory**

Memory

Solution

- Disable memory caching
- Require a thread ID whenever reading or writing memory

```
size_t Process::ReadMemory(addr_t addr, void *buf,
                           size_t size, Status &error,
                           lldb::tid_t tid) {
```

Threading

Problem

- LLDB expects a POSIX thread model
  - `NativeProcess/Thread/Register`
- Large number of threads with additional architectural hierarchy (e.g. warps/threads)
- **How to present usable debugging model while taking advantage of LLDB infrastructure?**

Threading

Solution

- Restrict vision of chip to a small slice (4 warps, 16 threads)
  - Architecturally relevant slice
  - User may select this slice dynamically
- Added support for hierarchy
  - Hierarchy-specific commands exposed to user
  - Debugger monitors whole chip opaquely to user

How can we contribute to the project?

- We haven't upstreamed anything yet
  - Difficult to untangle our work from private architecture
  - Difficult because the POSIX environment is so core to the existing code base
  - Intimidating as new contributors
- Thinking about how to extract and upstream parts of our work
  - Welcome suggestions from current contributors

Will we use LLDB in future projects?

Yes!