IWOCL 2026 · KEYNOTE

chipStar

CUDA/HIP on anything

OpenCL + SPIR-V as Portability Layer

Paulius Velesko
pvelesko@pglc.io

chipStar team · github.com/CHIP-SPV/chipStar

IWOCL 2026 · KEYNOTE

chipStar

CUDA/HIP on anything*

OpenCL + SPIR-V as Portability Layer

Paulius Velesko
pvelesko@pglc.io

chipStar team · github.com/CHIP-SPV/chipStar

From HIPCL to chipStar.

Two predecessor projects — HIP-on-OpenCL and HIP-on-Level-Zero — merged into a single runtime that does both.

2020 – 2021

HIPCL

HIP over OpenCL + SPIR-V

Tampere University / CPC. IWOCL 2020.

WHAT IT NEEDED

- OpenCL 2.x
- SVM
- `clCreateProgramWithIL`

~ 2021 – 2022

HIPLZ

HIP over Level Zero + SPIR-V

Derived from HIPCL. Aurora Intel GPUs.

WHAT IT ADDED

- Level Zero backend
- 35+ HIP tests on Gen9 / UHD 770
- Perf ≈ native OpenCL

Dec 2021 – present

chipStar

HIP / CUDA over OpenCL or Level Zero

formerly CHIP-SPV. CUDA + HIP + ★ → "run everywhere".

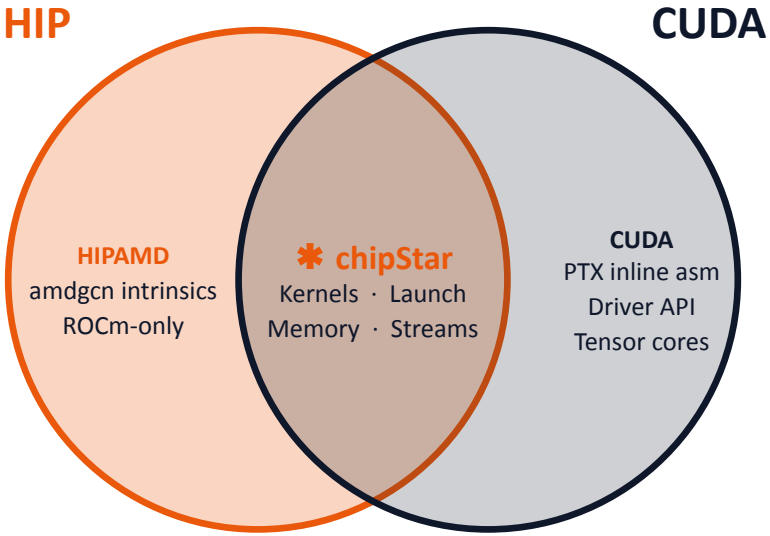
WHAT IT IS NOW

- OpenCL *and* Level Zero backends
- Apple Silicon via clvk + MoltenVK
- CUDA + HIP source compatibility

Two parallel attempts. One merged runtime. *chipStar is the synthesis — and the active project.*

HIP vs CUDA — and what we mean by “CUDA”.

This talk is about the kernel language API. Vendor math libraries (cuBLAS, cuFFT, cuDNN, ...) are a separate concern — addressed later.



> OUT OF SCOPE · `cuBLAS, cuFFT, cuDNN, cuRAND, cuSPARSE, NCCL, Thrust, ...` *vendor math libraries are a separate concern — covered in the libraries section.*

When this talk says “CUDA”, it means the language. *chipStar covers the overlap with HIP.*

THE DESIGN SPACE

Four categories of CUDA portability layer.

Each category is defined by what gets translated and what ultimately ships.

Category	What gets translated	What ships	Examples
Source-to-source translation	CUDA source ⟶ HIP / SYCL source	Translated source — you maintain it	HIPIFY, SYCLomatic
Source-level retargeting compiler	CUDA / HIP source ⟶ portable IR	One fat binary, finalized at load	chipStar, AdaptiveCpp PCUDA
Binary-compatible IR translation	PTX (from CUDA fat-bin) ⟶ device IR	Per-vendor runtime; original binary unchanged	ZLUDA
Native CUDA reimplementation	CUDA source ⟶ vendor ISA directly	Per-vendor native binary	SCALE

chipStar lives in the source-level retargeting category alongside AdaptiveCpp PCUDA.

THE DESIGN SPACE

Trade-offs of each portability model.

What each model gains, what it gives up. Why chipStar chose source-level retargeting.

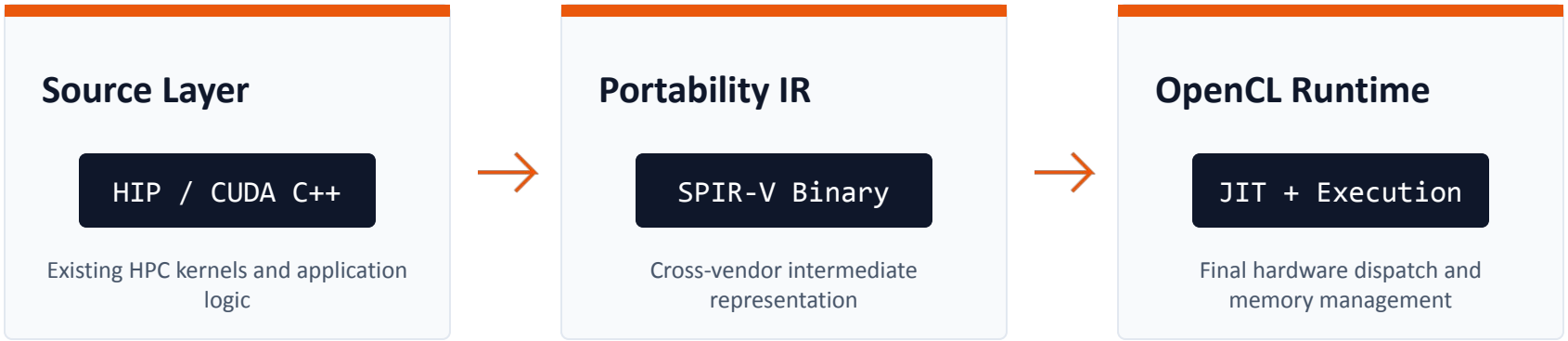
Category	Pro	Con	Examples
Source-to-source translation	Output is HIP / SYCL — native vendor tooling, libraries, debuggers all apply.	Maintenance of a separate backend.	*HIPIFY, SYCLomatic*
Source-level retargeting compiler	One codebase. One fat binary. Open standards underneath.	Bound to target IR expressiveness; vendor BLAS / FFT gap remains.	*chipStar, AdaptiveCpp PCUDA*
Binary-compatible IR translation	No source needed. Run unmodified CUDA binaries.	NVIDIA's EULA prohibits translation layers. AMD walked away in 2024.	*ZLUDA*
Native CUDA reimplementation	Native vendor ISA. Highest performance ceiling.	One implementation per vendor. Forever chasing CUDA's evolution.	*SCALE*

Why retargeting? *It's the only model that gives one codebase, one binary, and an IR no single vendor controls.*

ARCHITECTURE OVERVIEW

HIP Kernels to SPIR-V Execution Pipeline

Bridging CUDA/HIP source code to diverse hardware via open-standard intermediate representation



THE IR QUESTION

Choosing the IR: LLVM or SPIR-V.

Khronos has tried both LLVM-IR-as-distribution and a purpose-built IR. The first failed. The second is what chipStar emits.

ALTERNATIVE — AdaptiveCpp

LLVM IR (+ JIT)

2003 – present

Compile to LLVM bitcode. Ship a JIT compiler with the runtime to lower at dispatch time.

PROS

+ One IR across CPU and GPU
+ Runtime kernel modification (fusion, specialization)
+ Independent of driver SPIR-V quality

CONS

− LLVM bitcode unstable across LLVM versions
− Must ship LLVM JIT — heavy runtime
− Bypass driver-tuned compilation

DEPRECATED — abandoned 2015

SPIR (1.2 / 2.0)

2012 – 2015

Standardized LLVM IR (3.2 / 3.4) + Khronos metadata. OpenCL 1.2/2.0 extension.

PROS

+ First Khronos GPU IR standard
+ Drivers could consume directly

CONS

− Pinned to specific LLVM versions
− Non-LLVM vendors had to ship LLVM
− Metadata-heavy; backward compat painful
− Couldn't represent graphics shaders

★ chipStar's choice

SPIR-V

2015 – present

Purpose-built binary IR. Not LLVM-based. Used by OpenCL 2.1+, Vulkan, Level Zero.

PROS

+ Decoupled from LLVM versioning
+ Drivers consume directly — no JIT shipped
+ Stable, versioned spec

CONS

− Flavor split (OpenCL env vs Vulkan env)
− Driver SPIR-V quality varies

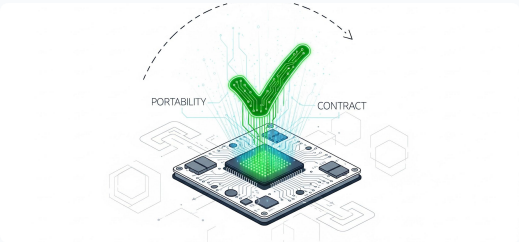
AdaptiveCpp ships a JIT runtime to dodge LLVM-IR instability. **chipStar avoids it: SPIR-V is stable, drivers consume it directly.**

The Industry Standard for Portable Parallelism

1. Open & Neutral

Khronos-governed standard

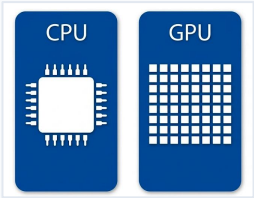
- Multi-vendor consortium ensures industry-wide alignment.
- Independent from specific compiler release cadences.



2. Validated Contract

Explicit capability system

- Decidable upfront: clean failure modes if unsupported.
- Forward-portable across evolving hardware generations.



3. Designed for GPUs

Native hardware alignment

- Structured control flow and explicit address spaces.
- Zero impedance mismatch: drivers consume code natively.

OpenCL Implementations w/SPIR-V

Intel Compute Runtime

Enables Intel CPUs & GPUs

- Intel Arc / Xe Discrete
- UHD / Iris Xe iGPU
- Core / Xeon CPUs

rusticl (Mesa)

Open-stack OpenCL on Gallium

- AMD Radeon (RDNA/GCN)
- NVIDIA RTX / GeForce
- Supported via NVK/RadeonSI

clvk

OpenCL on Vulkan 1.1+

- Apple A/M-Series (Metal)
- Qualcomm Adreno
- Arm Mali & Universal Vulkan

PoCL

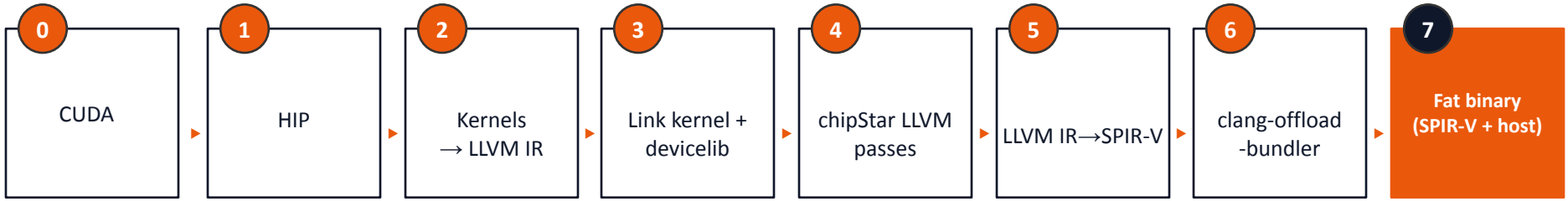
Portable OpenCL for X

- x86-64 (Intel / AMD)
- AArch64 (Graviton/Apple)
- RISC-V (SiFive/T-Head)
- Intel GPUs via Level Zero

SECTION

Toolchain

Upstream Clang/LLVM + out-of-tree passes.



Supported in LLVM & CMake

chipStar supported in **LLVM 22.0** and **CMake 4.3.0**

LLVM passes

Out-of-tree, but apply to standard LLVM IR.

Offload bundle

Embedded as a magic global in the host binary.

What chipStar's LLVM passes actually do.

Two jobs: bridge HIP/CUDA features that OpenCL doesn't have, and work around bugs in the SPIRV-LLVM translator.

BRIDGE *synthesize features OpenCL doesn't expose*

`HipDynMem`	extern __shared__ → kernel argument
`HipPrintf`	CUDA printf → OpenCL printf
`HipAbort`	kernel abort() → flag + early returns
`HipGlobalVariables`	__device__ globals → i64 slot + 3 kernels
`HipTextureLowering`	HIP textures → SPIR-V image ops
`HipWarps`	force subgroup size = warp size
`HipKernelArgSpiller`	big arg lists → side buffer

WORKAROUND *shape IR so SPIRV-LLVM accepts it*

`HipFixOpenCLMD`	force opencl.ocl.version = 2.0
`HipDefrost`	strip freeze instructions
`HipLowerSwitch`	switch on i4 / i33 → std width
`HipLowerMemset`	llvm.memset → explicit loop
`HipLowerZeroLenArrays`	T[0] types → legal types
`HipPromoteInts`	i33 / i56 → i64
`HipStripUsedIntrinsics`	drop llvm.used

RUNTIME SUPPORT *emit metadata the chipStar runtime uses at module load*

`HipIGBADetector`	scan for ptr loads / inttoptr → __chip_module_has_no_IGBAs flag
`HipEmitLoweredNames`	build symbol → mangled name table for HIP RTC

Each pass exists because something in HIP doesn't map to SPIR-V. *When the API is missing, synthesize it in IR.*

INSIDE THE BLACK BOX

Six passes hit by ten lines.

Every CUDA / HIP construct that can't go straight to SPIR-V is rewritten by a dedicated pass.

kernel.hip

```
__constant__ float kCoeff; (1)

struct Args { float* data; int n; }; (2)

__global__ void scale(Args a) {
  extern __shared__ float buf[]; (3)
  int tid = threadIdx.x;
  if (tid >= a.n) abort(); (4)
  buf[tid] = a.data[tid] * kCoeff; (2)
  if (__shfl_xor(buf[tid], 1) > 0) (5)
    printf("tid=%d val=%f\n", tid, buf[tid]); (6)
}
```

PASSES TRIGGERED

1. **HipGlobalVariables**
   host R/W of __constant__ via SVM
2. **HipIGBADetector**
   ptr-in-struct + load-of-ptr → flag stays false
3. **HipDynMem**
   extern __shared__ becomes __local arg
4. **HipAbort**
   rewrites callers, sets host-readable flag
5. **HipWarps**
   annotates kernel with reqd_subgroup_size
6. **HipPrintf**
   fmt → constant AS, lowered to OpenCL printf

Six passes — *none of these constructs reaches SPIR-V untouched.*

SECTION

Memory Spaces

MEMORY MODEL

How HIP pointers fit on top of OpenCL — auto-selected fallback chain.

Strategy	Extension	Why we want it	If absent
1. Intel USM	`cl_intel_unified_shared_memory`	Closest match to CUDA unified memory; host/device/shared allocations.	Falls back to SVM.
2. Fine SVM	`CL_DEVICE_SVM_FINE_GRAIN_SYSTEM`	CPU/GPU pointer sharing without explicit transfers.	Falls back to fine grain SVM
3. Coarse SVM	`CL_DEVICE_SVM_COARSE_GRAIN`	CPU/GPU pointer sharing + Map/Unmap.	Falls back to device buffer address
4. Buffer Device Address	`cl_ext_buffer_device_address`	Raw opaque device. Cleanest map to CUDA device pointers.	:(

SECTION

Device Library

Three sources, in priority order.

chipStar's device library is an integrator. Every device-side function is OpenCL, OCML, or — only when neither covers it — ours.

01 OpenCL builtin

Vendor-tuned, often one instruction from silicon. **Preferred whenever it exists.** Standard math (sin, cos, sqrt, atan2 …), atomics, subgroup ops.

02 OCML (ROCm-Device-Libs)

AMD's open-source math library, vendored in-tree. Production implementations of math OpenCL doesn't cover — Bessel functions, erfc inverse, normal CDF.

03 Custom (bitcode/devicelib.cl)

~1,900 lines of OpenCL-C for what nobody else implements. Warp primitives, textures, abort() flag, integer-norm, jn / yn (OCML's are AMD-only).

FROM include/hip/devicelib/

```
// sp_math.hh:44   — Tier 1
extern "C++" __device__ float
acos(float x);        // OpenCL

// dp_math.hh:232  — Tier 2
extern "C" __device__ double
__ocml_j0_f64(double x);  // OCML

// dp_math.hh:242  — Tier 3
extern "C" __device__ double
__chip_jn_f64(int n, double x);
                                  // Custom
```

One principle — *prefer the implementation closest to the hardware. Write our own only when nobody else has.*

DEVICELIB · MATH PRECISION

Why correctness-first matters.

OpenCL native_* functions have no ULP spec. On a CPU device, the same call can be off by 1000×. From paper Tables 1–2.

Function	Intel A770	AMD gfx906	Intel i9 CPU	Takeaway
`cos (OpenCL std)`	1	**1**	2	**spec ≤4 ULP — all pass**
`exp (OpenCL std)`	2	1	1	**spec ≤3 ULP — all pass**
`log (OpenCL std)`	1	2	1	**spec ≤3 ULP — all pass**
`native_cos`	27.5	6.07	1085	**no spec — CPU unusable**
`native_exp`	0.84	0.67	475	no spec — CPU unusable
`native_log`	0.36	0.46	155	no spec — CPU unusable
`CUDA spec`	≤1 ULP	—	—	**What apps assume**
`HIP 6.1 spec`	unspec	unspec	unspec	**Specced in 6.3, still ≠ CUDA**
`chipStar choice`	standard	standard	standard	**Correctness over speed**

Standard OpenCL is portable and predictable. *native_* is not. We map regular and intrinsic to the same standard built-ins.*

rtdevlib: capability swap at JIT time.

When a function has a fast and slow form, pick at link time based on what the driver advertises.

WHAT'S IN IT

A small set of OpenCL-C modules — currently atomic-add float / double, atomic min/max float, ballot — each compiled to a SPIR-V module and embedded into the chipStar library.

HOW IT WORKS

On JIT load, the runtime checks driver capabilities and appends the matching variant. `atomicAdd<float>` gets the native module if `cl_ext_float_atomics` is advertised; otherwise the CAS-loop emulation.

THE STRATEGIC PAYOFF

When a driver ships a new extension — float atomics, generic-AS atomics, better subgroup ops — **every chipStar binary on that platform inherits it on the next launch**. No recompile.

bitcode/CMakeLists.txt:131-159 · CHIPBackendOpenCL.cc:1223

RUNTIME PICKS BASED ON…

```
atomicAdd(float*, float)
if  cl_ext_float_atomics
  →  atomicAddFloat_native
else  →  atomicAddFloat_emulation

atomicAdd(double*, double)
if  cl_ext_float_atomics  (fp64)
  →  atomicAddDouble_native
else  →  atomicAddDouble_emulation

__ballot(int)
if  subgroup ballot extension
  →  ballot_native
else  →  (emulated in devicelib.cl)
```

Correctness first, simplicity second. *Optimization is end-user driven — we tune when a workload bites someone.*

SECTION

Runtime

RUNTIME · BEFORE MAIN

How HIP code finds its kernels.

Every HIP binary runs a small ceremony before main(). Clang emits constructors that hand the SPIR-V module and every kernel symbol to the chipStar runtime.

WHAT CLANG EMITS

auto-generated module constructor for every HIP TU

```
static void __attribute__((constructor))
__hip_module_ctor() {
  if (!Handle)
    Handle = __hipRegisterFatBinary(...);
  __hip_register_globals(Handle);
  atexit(__hip_module_dtor);
}

static void __hip_register_globals(void** H) {
  // one per __global__ definition
  __hipRegisterFunction(H, ...);
  ...
  // one per __device__ / __constant__
  __hipRegisterVar(H, ...);
  ...
}
```

WHAT CHIPSTAR DOES

each call binds a piece of state, no GPU work yet

__hipRegisterFatBinary

Extract SPIR-V from the embedded fatbin. Register it in SPVRegister. Return module handle.

No driver compile, no backend init — by design.

__hipRegisterFunction

Bind a host-side function pointer → SPIR-V kernel name.

Later, kernel<<<...>>>(args) does HostPtrLookup → SPIR-V symbol.

__hipRegisterVar

Bind a host-side variable pointer → __device__ symbol name + size.

Wires up the slot the HipGlobalVariables pass created.

Faking the null stream with explicit event chains.

THE HIP CONTRACT

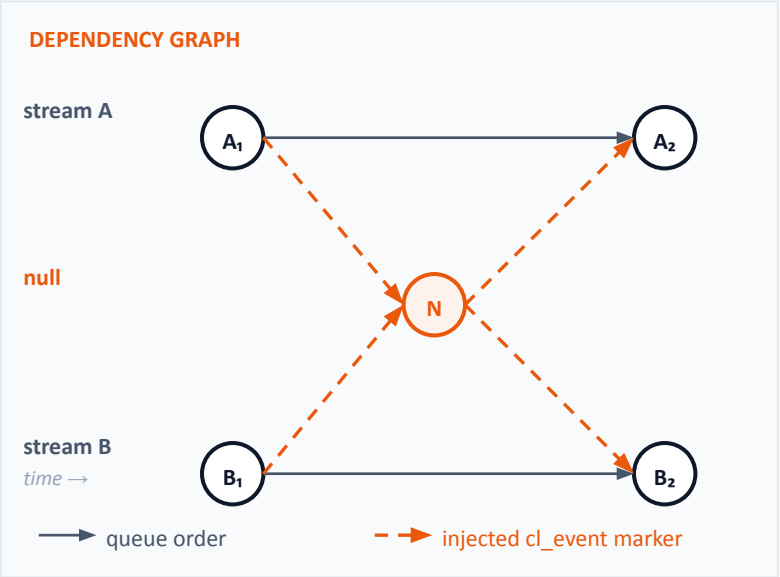
In-order within a stream. Null stream synchronizes with all other streams — full barrier on entry and exit.

WHAT OPENCL GIVES YOU

Per-queue ordering and explicit `cl_event` wait-lists. **No default queue. No cross-queue barrier primitive.**

HOW CHIPSTAR BRIDGES IT

Each null-stream submission walks every live blocking queue and enqueues a `clEnqueueMarkerWithWaitList` in each. Those `cl_events` become the null op's wait-list. Symmetric on the way out.



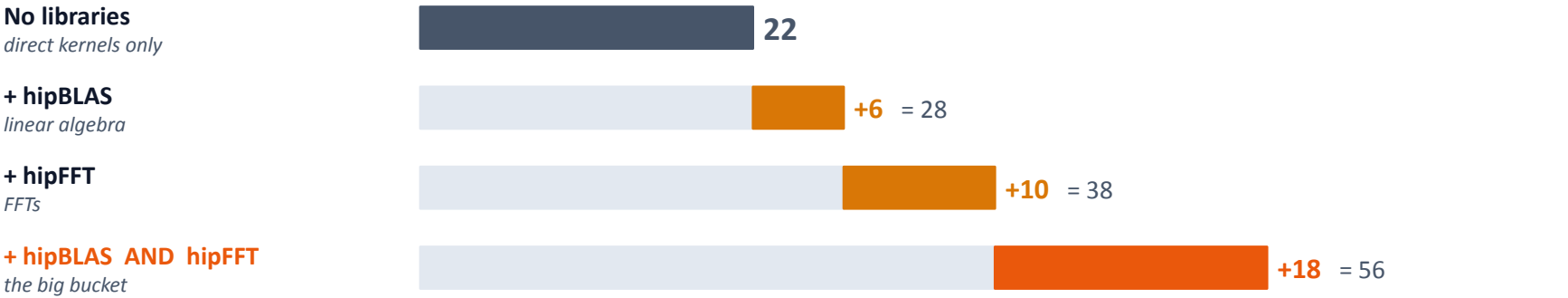
Abstraction leaking — *Every null-stream op pays O(N) in the number of live blocking streams: one marker enqueued per queue, each tracked and kept alive until its waiters complete.*

SECTION

Library Coverage

How many apps each library unlocks.

Real HPC and scientific applications, grouped by which CUDA libraries they call. Each bar is what chipStar reaches as we add library coverage.



THE BIG BUCKET — apps that need both `hipBLAS` AND `hipFFT`

GROMACS · OpenMM · AMBER · RELION · Quantum ESPRESSO · VASP · ABINIT · Octopus · BerkeleyGW · TeraChem · Tinker-HP · BigDFT · GPAW · FLEUR · CryoSPARC · Parabricks · ICON · IFS

Two libraries together unlock the bulk of computational science.

LIBRARIES

What's already supported.

Pure-HIP / header-only libraries, ported once, run everywhere chipStar runs.

- **`rocPRIM`**

  Parallel primitives — scan, reduce, sort, partition.

- **`hipCUB`**

  CUB-style block- and warp-level primitives.

- **`rocThrust`**

  Thrust parallel algorithms over HIP.

- **`rocRAND / hipRAND`**

  RNG kernels — pseudo and quasi-random.

- **`rocSPARSE / hipSPARSE`**

  Sparse matrix-vector and matrix-matrix ops.

- **`hipMM`**

  Memory manager — RMM port for HIP.

- **`hipFTT`**

  FFT support for GPUs via VkFFT.

- **`hipBLAS/hipFFT/hipSOLVER`**

  Intel only via MLK Shim Layer.

All ported to chipStar's CHIP-SPV org on GitHub. *These libraries don't need vendor-specific tuning to work — they're HIP all the way down.*

LIBRARIES · THE GAP

GPU BLAS / FFT / SOLVER — only four vendors ship one.

Vendor-tuned assembly, not portable HIP. Most GPU vendors don't have one at all.

Vendor	Math libraries	GPU library?
Intel	MKL (BLAS, FFT, LAPACK, sparse, RNG)	yes (CPU + GPU, oneAPI)
AMD	rocBLAS, rocFFT, rocSOLVER, rocSPARSE, rocRAND (ROCm)	yes (AMD GPU)
NVIDIA	cuBLAS, cuFFT, cuSOLVER, cuSPARSE, cuRAND (CUDA / HPC SDK)	yes (NVIDIA GPU)
Moore Threads	muBLAS, muFFT, muThrust (MUSA Toolkit)	yes (MT GPU)
Arm	*ArmPL (BLAS / LAPACK / FFT, AArch64 CPU) · ACL is ML-only*	**no (Mali: no BLAS/FFT)**
Qualcomm	*Adreno SDK + QNN (inference). No BLAS / FFT / SOLVER.*	**no**
Broadcom	*No compute SDK. VideoCore: community-only OpenCL.*	**no**
RISC-V	*no equivalent yet.*	*open question*

Where chipStar sits today. On Intel, hipBLAS / hipFFT / hipSOLVER route through `MKL` via the `H4I-MKLShim` bridge — Intel-only.

If you want to run HPC on a new architecture, **someone has to provide the library.** *RISC-V is the open question.*

SECTION

API Coverage

COVERAGE · HIP API SURFACE

224 of 319 HIP API entry points.

Counted from chipStar's CHIPBindings.cc — every public entry point with an UNIMPLEMENTED stub is in the missing set.

HEADLINE COVERAGE

70.2%

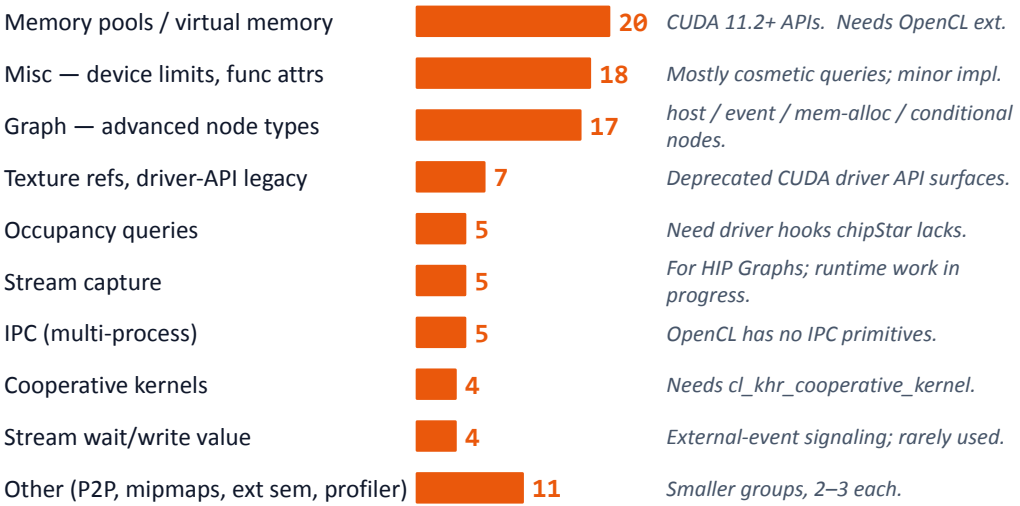
of HIP API entry points implemented

224 implemented

95 stubbed (UNIMPLEMENTED)

WHAT'S IN THE MISSING 95

Each API counted exactly once. Largest categories first.



The missing 30% clusters cleanly. *Most of it is recent CUDA APIs and features blocked on OpenCL spec or runtime work.*

OPEN ASKS · KHRONOS

Blockers

Three CUDA features chipStar doesn't support today. Each needs OpenCL spec or driver work.

Cooperative groups

Grid-wide synchronization. Co-residency guarantee for all workgroups. Level Zero has the host launch; device-side primitive missing.

Spec	needs SPIR-V ext
Vendor	n/a
Driver	none implemented

NEEDS SPIR-V EXT

THE ASK

Define a Grid execution scope for OpControlBarrier in SPIR-V. One extension benefits both OpenCL and Level Zero.

HIP Graphs

Pre-recorded command graphs. Stream capture, replay, conditional nodes.

Spec	cl_khr_command_buffer
Vendor	ARM Mali G52, PoCL
Driver	Intel CPU; PoCL WIP

WAITING ON VENDORS

THE ASK

Broader vendor adoption — extension is Public, but most GPU drivers don't ship it yet.

Group barriers with early exit

CUDA group barriers don't count exited threads. OpenCL says undefined behavior — many drivers deadlock.

Spec	cl_ext_alive_only_barrier (Draft)
Vendor	none committed
Driver	none yet

DRAFT · HIT IN GAMESS PORT

THE ASK

Promote cl_ext_alive_only_barrier from Draft. Required a refactor in GAMESS-GPU-HF, libCEED

These aren't chipStar's TODO list. **They're three concrete asks the OpenCL community could deliver.**

SECTION

Hardware Coverage

PORTABILITY · HARDWARE COVERAGE

CUDA/HIP On Anything

Same SPIR-V

Hardware	Vendor	OpenCL runtime
`Intel CPU/GPU`	Intel	Intel Compute Runtime
`NVIDIA GPU`	NVIDIA	rusticl/zink
`AMD GPU`	AMD	rusticl/radeonsi
`RISC-V CPU`	StarFive	PoCL
`PowerVR GPU`	Imagination	Imagination proprietary
`ARM CPU`	ARM	PoCL
`ARM GPU`	ARM	ARM proprietary
`M4 CPU`	Apple	PoCL

CASE STUDY · PoCL · APPLE M4 CPU

CUDA on M4 CPU

chipStar → PoCL → Apple M4 CPU. macOS now a first-class target.

UNIT TEST PARITY

Pass rate vs Intel CPU + Linux baseline running PoCL.

✓ **95%**
unit-test pass rate · within margin of Intel CPU + Linux

✓ **HIP + macOS**
MacOS changes already merged upstream, part of LLVM 23 release.

✓ **PoCL**
Portable OpenCL CPU implementation. Same one used on Linux x86 / RISC-V / Arm.

CASE STUDY · CLVK

Vulkan reaches the GPUs OpenCL doesn't.

Apple, mobile, and embedded ship Vulkan compute drivers — but no usable OpenCL ICDs. clvk bridges chipStar to all of them.

01 Apple GPU Silicon (A/M-series)

Vulkan reaches via MoltenVK over Metal — first time HIP on M1 / M2 / M3 / M4.

02 Arm Mali

Vulkan mandatory on Android since 2017. OpenCL support is variable across vendors and SoC generations.

03 Qualcomm Adreno

Vulkan ships on every Snapdragon. OpenCL is closed-source and gated to select OEMs.

04 Mesa Vulkan drivers

v3d (Raspberry Pi), panvk (Mali), turnip (Adreno). Open-source compute on community hardware.

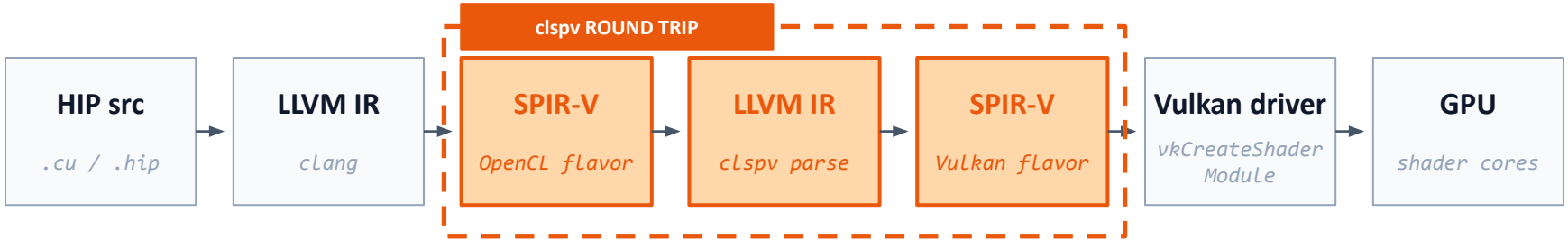
Three years ago: HIP on Intel. **Today through clvk: HIP on every modern GPU class.**

chipStar — IWOCL keynote

CASE STUDY · CLVK

The clvk path takes the long way around.

The default route round-trips our SPIR-V through LLVM IR — to come back to SPIR-V.



THE COST

- Subprocess invocation per kernel — clspv ships its own LLVM (~hundreds of MB). **Multi-second latency.**
- 31 distinct invalid-Vulkan-SPIR-V failures observed on chipStar's HIP test suite.
- Conceptually wasteful: we already had SPIR-V. clspv decompiles to LLVM IR and recompiles back.

Right tool for OpenCL-C source. **Wrong tool when we already have SPIR-V.**

CASE STUDY · CLVK · APPLE M4 GPU

CUDA on Apple M-Series GPU

chipStar⟶ clvk ⟶ MoltenVK ⟶ Metal on M-series silicon.

VERIFIED END-TO-END

Output contains explicit PASS marker, no errors anywhere in the run.

✓ **haccmk-cuda**
Hardware-Accelerated Cosmology mini-kernel · LCF benchmark

✓ **floydwarshall-cuda**
Graph all-pairs shortest paths

✓ **hmm-cuda**
Hidden Markov Model — Viterbi decoding

✓ **overlap-cuda**
Stream-overlap throughput micro-benchmark

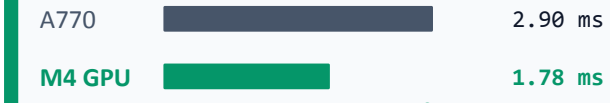
+ 9 partial-PASS *— printed PASS for the part that ran before a crash*

PERFORMANCE · vs ARC A770

Same CUDA source, Kernel-avg time.

haccmk

1000 iters, kernel avg



M4 is 1.6× faster than A770

floydwarshall

1024² graph, 100 iters, kernel avg

A770 is 3.6× faster — work to do

CASE STUDY · CUDA ON iOS

chipStar + clvk + PoCL Remote = CUDA on iOS

Compile on the Mac. Dispatch over TCP. Run on the A16. The phone is a network-attached GPU.

375 GFLOPS *fp32 1024² matmul on iPhone 14 Pro (A16)*

YOUR MAC

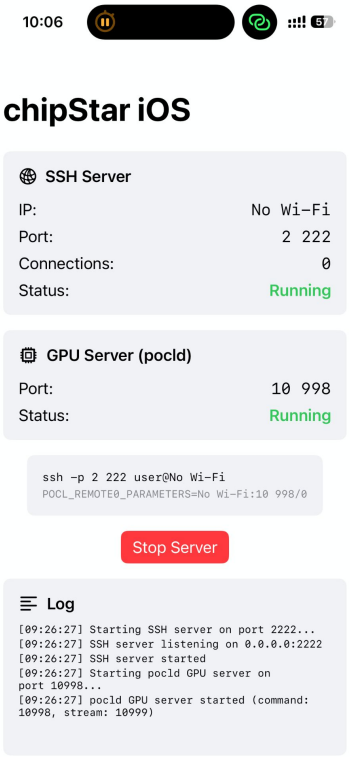
compilation + runtime + transport

- `.cu source` — *ordinary CUDA kernels*
- **hipcc** — *chipStar's clang + LLVM passes*
- `host .o + embedded SPIR-V` — *kernels baked into the binary*
- **libCHIP** — *CUDA/HIP runtime ⟶ OpenCL calls*
- **PoCL libOpenCL.dylib** — *ICD with remote backend*

TCP

10998 / 10999

Wi-Fi · USB



CASE STUDY · CUDA WHERE IT SHOULDN'T BE

N-Body Simulation on Samsung Smart Fridge

Runs cool

40 GFLOPS *fp32 1024² matmul on Samsung Exynos 3250*

YOUR HOME FRIDGE

Snacks, Drinks, Compute

- `.cu source` *ordinary CUDA kernels*
- `hipcc` *chipStar's clang + LLVM passes*
- `host .o + embedded SPIR-V` *kernels baked into the binary*
- `libCHIP` *CUDA/HIP runtime → OpenCL calls*
- `PoCL libOpenCL.dylib` *ICD with remote backend*

TCP

10998 / 10999

Wi-Fi · USB

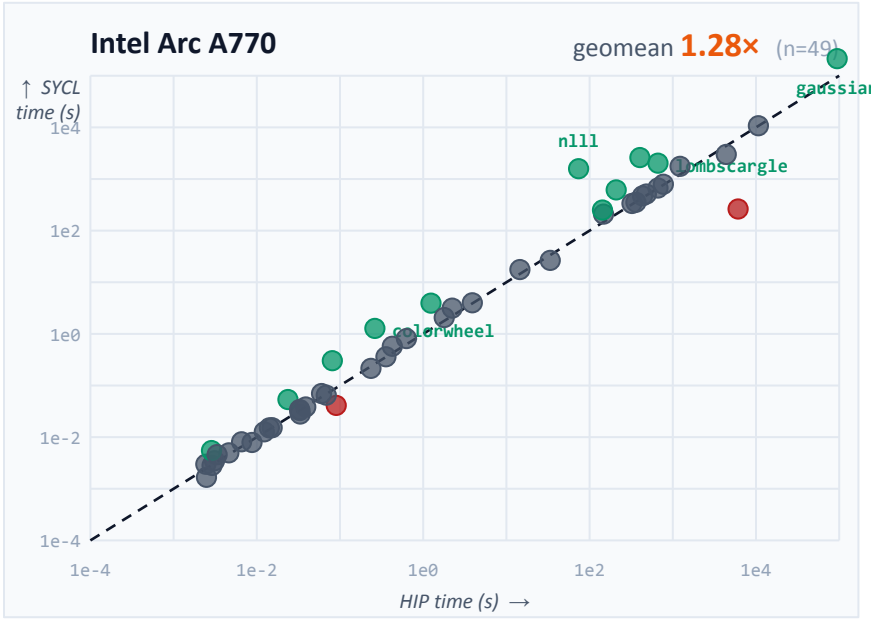
SECTION

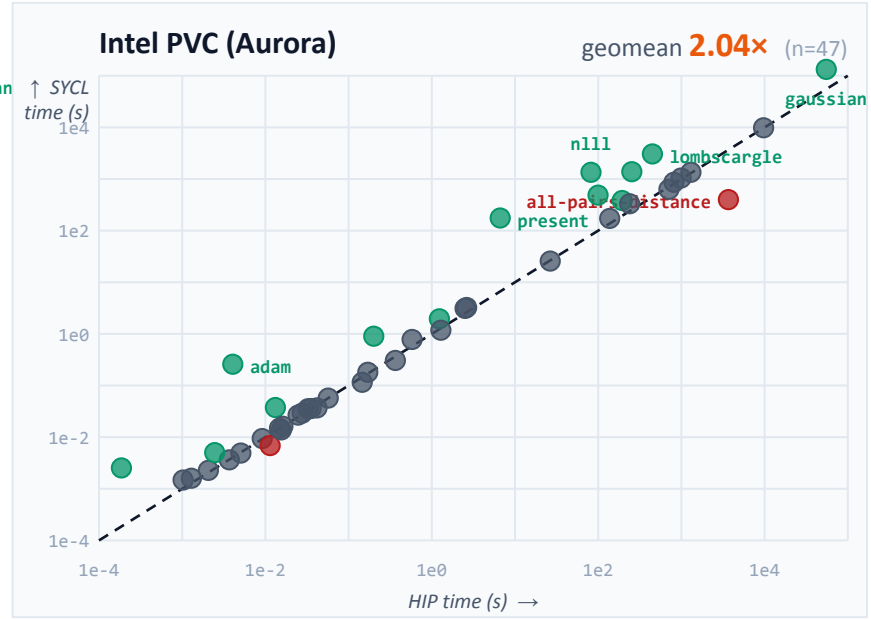
Performance

CASE STUDY · HECBENCH

chipStar vs SYCL on the same hardware.

HeCBench: each point is one benchmark on the same hardware. Diagonal is parity. Above the line: chipStar HIP path is faster than SYCL.





chipStar ≥1.5× faster · within ±50% · SYCL ≥1.5× faster

A770: paper-flag config (-cl-fast-relaxed-math, OpenCL backend) · PVC: no-fast-math (Level Zero)

Most points sit above the diagonal. *On geomean, chipStar's HIP path beats SYCL on the same hardware — 1.28× on A770, 2.04× on PVC.*

PERFORMANCE · CUDA SDK COMPARISON

chipStar vs NVIDIA CUDA SDK.

Same CUDA source. chipStar fat binary on rusticl/zink — OpenCL on top of NVIDIA's Vulkan driver. RTX 3060, strict math. Old data from chipStar v1.2.1

GEOMEAN OVERHEAD

1.01×

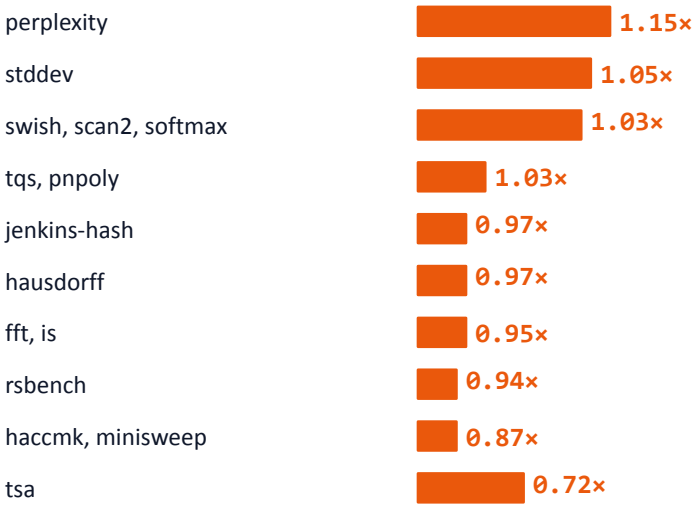
vs NVIDIA CUDA SDK 12.4

51 HeCBench benchmarks compared

36 finished within ±3% of CUDA SDK

15 OUTLIERS (≥3% DIFFERENCE)

Speedup of chipStar over CUDA SDK. Above 1× = chipStar faster.



chipStar vs AMD ROCm.

Same HIP source, same Radeon Pro VII GPU. AMD's OpenCL doesn't accept SPIR-V — chipStar runs on rusticl/radeonsi instead. Old data from chipStar v1.2.1

GEOMEAN

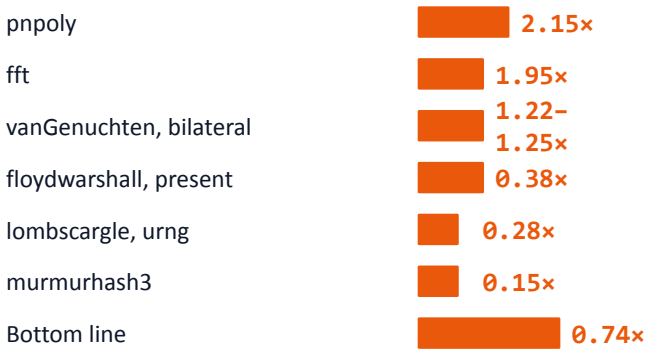
0.74×

vs AMD ROCm 6.2.4 on Radeon Pro VII

32 HeCBench benchmarks

~0% chipStar overhead (Figure 3)

WHERE THE 26% GAP COMES FROM



SECTION

Enabling Applications

PORTING CHALLENGES

It isn't the syntax we can't translate.

It's the silicon-level guarantees CUDA programmers learned to rely on that no portable spec promises.

01 Lock-step warp execution

Hardware-guaranteed on AMD and pre-Volta NVIDIA. **No SPIR-V extension says so**. Code that drops intra-warp sync may race or produce wrong answers.

libCEED.

02 __syncthreads divergence + early-exit

Idiomatic `if (tid >= n) return;` before a later barrier is UB but works on NVIDIA. OpenCL drivers are stricter — workgroup hangs.

libCEED, GAMESS

03 Hard-coded warp/wave width

Algorithms baked for `warpSize == 32` or 64. HipWarps pins the size; the algorithm itself may still be wrong for the chosen width.

04 Vendor library coverage

rocPRIM / hipCUB / rocThrust run everywhere. hipBLAS / hipFFT / hipSOLVER currently route through MKL — **Intel-only**. Biggest practical adoption blocker.

The pattern — *three of the four are semantic, not syntactic. chipStar bridges what the spec lets it bridge; the rest is on Khronos to standardize.*

APPLICATIONS · VERIFIED

Verified Applications

Production scientific codes running through chipStar with verified-correct results.

OpenMM

molecular dynamics

libCEED

high-order finite element

miniBUDE

drug docking

OCCA / BK5

spectral element

LULESH

hydrodynamics

XSBench

Monte Carlo neutron transport

OpenSn

discrete-ordinates radiation transport

LAMMPS

molecular dynamics

SECTION

Case Studies

Case Study: OpenMM

OpenMM: molecular dynamics for biology.

OPENMM ON AURORA PVC THROUGH chipStar

The application

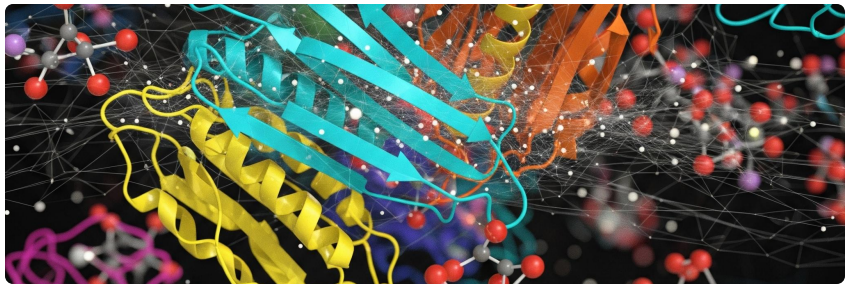
Open-source toolkit for molecular dynamics, widely used in computational biology and drug discovery. Maintains its own hardware backends in C++ — Reference, CPU, CUDA, OpenCL, and HIP — so the same simulation runs everywhere.

- Powers Folding@Home, the largest distributed biomolecular MD project.
- ~50,000 lines of HIP kernels — force fields, PME, integrators, FFTs.
- Already had OpenCL and HIP backends — clean baseline for chipStar comparison.

What chipStar had to provide

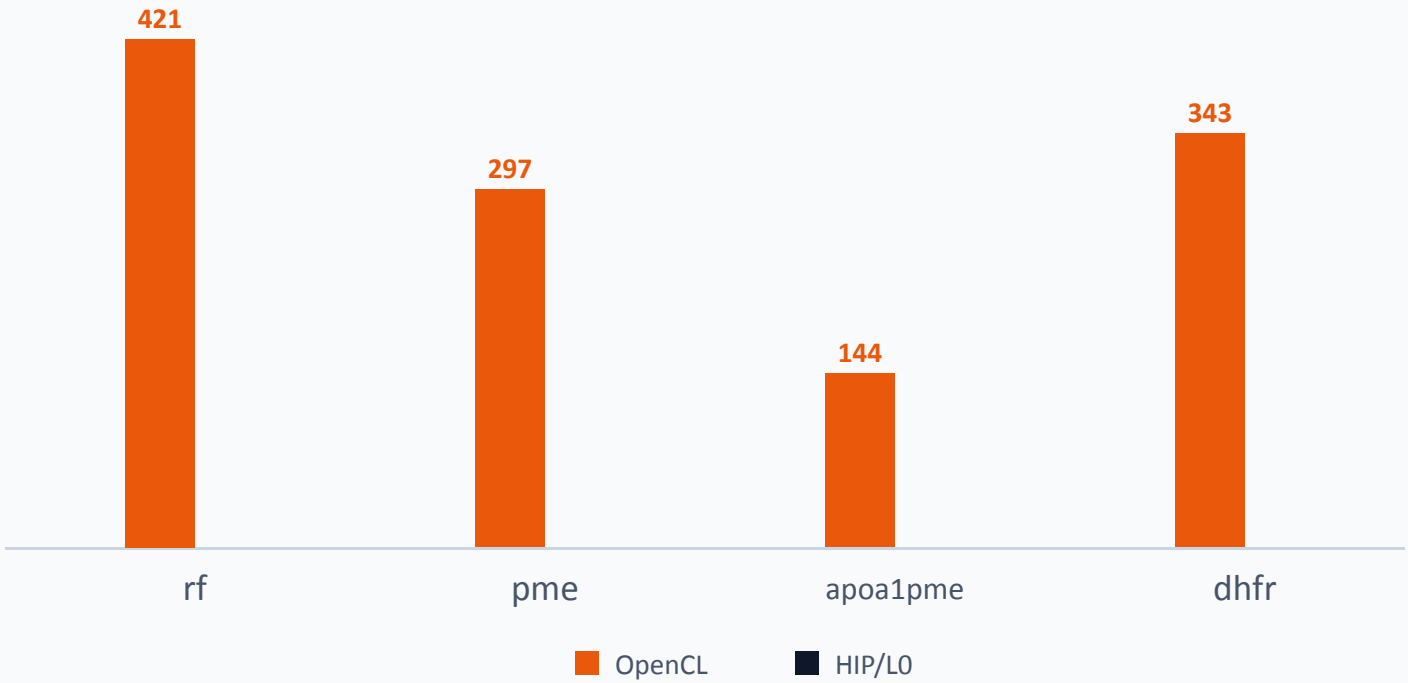
Mostly a recompile. Existing OpenCL and HIP backends meant no kernel rewrites — just a build-system path through chipStar:

- Update CMake files to be aware of chipStar library names.
- Rewrite amdgcn instrinsics



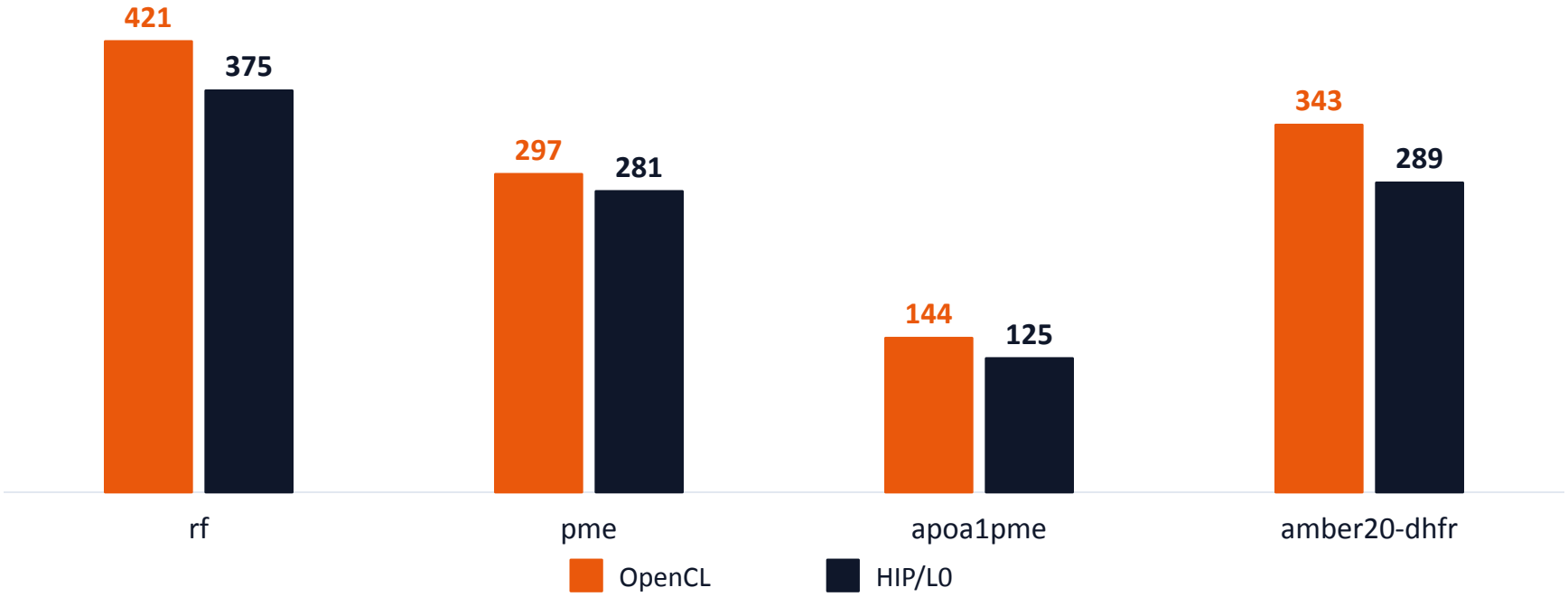
Free Lunch

OPENCL VS. HIP/L0 ON AURORA PVC, NO CODE CHANGES



Before the SLM refactor.

__builtin_amdgcn_ds_bpermute → __shfl, hip-config.cmake



84–95% of OpenCL across the suite.

Case Study: OpenMM

Performance Portability

ANALYSIS: OPENMM OPENCL VS. HIP/L0 ON AURORA PVC

CRITICAL GAP: The most expensive kernel is 2.8x slower on HIP compared to OpenCL.

OpenCL: Xe-HPC Native Alignment

- **SLM-BASED CALCULATION:** Optimized for inter-thread primitives using Shared Local Memory.
- **Vendor Tuning:** Accumulated Intel-specific tuning over time (e.g., msinclair fork).
- **Explicit Controls:** Manually sets simdWidth=32 and forceThreadBlockSize=128.

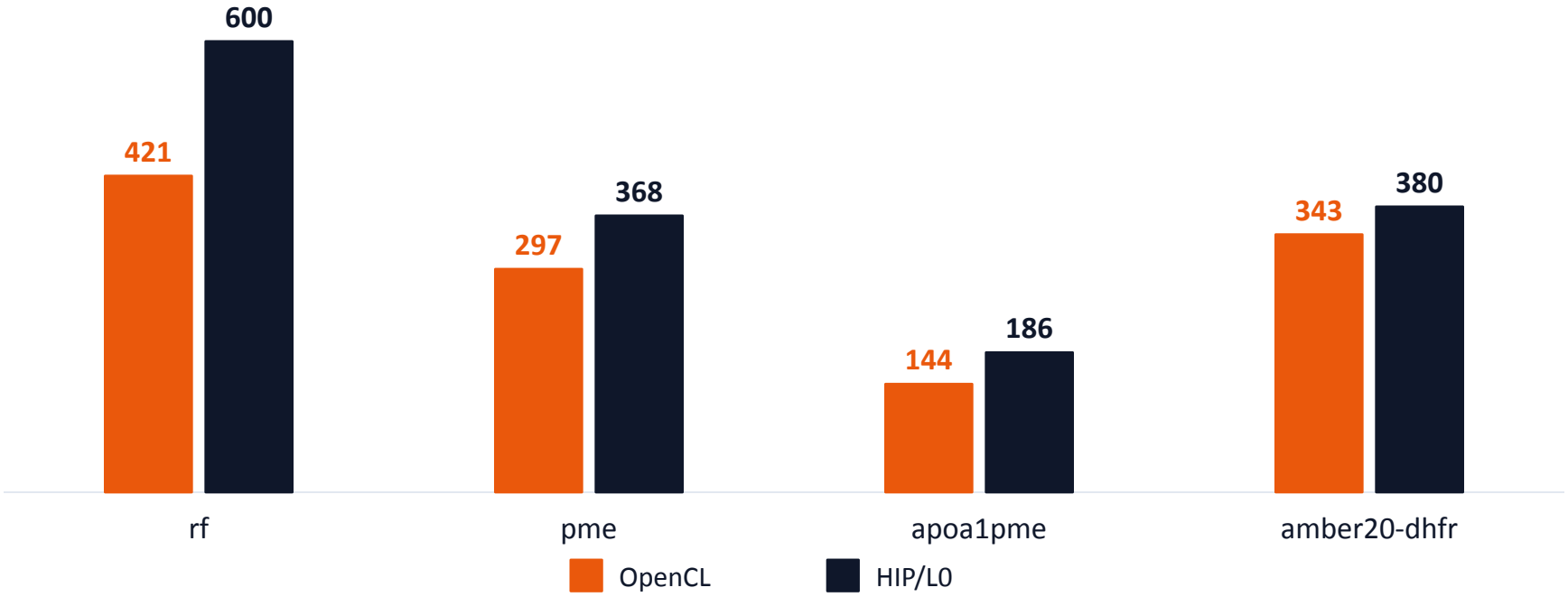
HIP: The Portability Tax

- **SHFL-BASED CALCULATION:** Relies on `__shfl` as the primary exchange mechanism.
- **Architecture Mismatch:** Designed for AMD/NVIDIA where warp shuffle beats LDS.
- **Translation Overhead:** Assumptions don't map to PVC via chipStar → SPIR-V → IGC.

Conclusion: Performance data shows each kernel is well-optimized for its intended target; OpenCL simply happens to be a better fit for Xe-HPC today.

After SLM-backed warp shuffle.

Lower warp-shuffle to shared local memory instead of subgroup ops. HIP overtakes OpenCL.



HIP/L0 now beats OpenCL on every benchmark. *+42%, +24%, +29%, +11%. Same code through chipStar.*

CASE STUDY · OPENMM

Per-kernel breakdown.

Per-kernel device time on Aurora PVC (µs, lower is better). HIP A is the pre-SLM baseline; HIP B is after SLM-backed warp shuffle.

Kernel	HIP Base	HIP + SLM	OpenCL	B vs OpenCL
`computeNonbonded`	403	**110**	142	**HIP 1.3× faster**
`findBlocksWithInteractions`	71	72	326	**HIP 4.5× faster**
`gridSpreadCharge`	38	39	91	**HIP 2.3× faster**
`gridInterpolateForce`	29	30	46	**HIP 1.5× faster**
`VkFFT_main`	9.6	9.7	10.1	≈ equal
`assignElementsToBuckets`	56	54	56	≈ equal
`computeBondedForces`	28	32	29	**HIP 1.1× slower**
`sortShortList(2)`	76	77	70	**HIP 1.1× slower**
`findBlockBounds`	33	33	15	**HIP 2.2× slower**

computeNonbonded is the SLM win (4× faster). *findBlockBounds is the open work (2.2× slower).*

Case Study: libCEED

libCEED: high-order finite element operators.

PORTING libCEED TO chipStar — AURORA PVC

The application

Open-source library for high-order finite element operators in HPC. Same C source runs on CPU, CUDA, HIP, SYCL, and MAGMA backends — a real-world test of HIP→OpenCL/L0 portability.

- Cornerstone of DOE ECP CEED ecosystem.
- Used by MFEM, Nek5000, PETSc downstream.
- JIT-specialized kernels per element / quadrature.
- ex1/ex2/ex3 volume + surface FEM benchmarks.

What chipStar had to provide

Two upstream PRs to libCEED + chipStar-side enabling work:

- PR #1942 — Makefile chipStar detection, SYCL co-link, flag filtering.
- PR #1950 — kernel fix: padded loop bound for uniform __syncthreads.
- chipStar: HipWarps pass + CHIP_FORCE_SUBGROUP_SIZE; hipBLAShip for macOS.

CASE STUDY · libCEED · PERFORMANCE

chipStar HIP beats SYCL on Intel.

Aurora PVC · 50M unknowns · 200 timed iters. Same C source — chipStar HIP path vs Intel SYCL.

Backend	ex1 (s)	ex2 (s)	ex3 (s)	vs SYCL gen
chipStar hip/gen · L0	4.6	**7.6**	9.1	**1.21–1.13× faster**
chipStar hip/gen · OpenCL	4.7	7.6	9.2	**≈ same as L0**
Intel SYCL/gen	5.8	8.6	9.7	**baseline**
— shared backends —				
chipStar hip/shared · L0	5.6	11.4	12.2	mixed
Intel SYCL/shared	6.9	10.6	12.4	—
— ref backends —				
chipStar hip/ref · L0	7.3	14.4	16.7	**reference path**
Intel SYCL/ref	7.0	12.0	14.3	—

chipStar HIP-gen beats Intel SYCL by 17–25% across ex1/ex2/ex3. *L0 and OpenCL backends within noise.*

Case Study: LAMMPS

LAMMPS: chipStar at 0.25x of OpenCL

INTEL ARC A770 · 32K-ATOM LJ · vs PURE-OPENCL LAMMPS

What chipStar had to provide

Three chipStar fixes (branch lammps-fixes) + LAMMPS-side downstream patches:

- cmake hip-config-in: pin hipcc path so consumer-side custom commands resolve.
- chipFFT linked in for AMOEBA FFT path; cufft → hipFFT shim downstream.

What we found

Builds and runs to completion. Dynamics are right. But:

- 584/613 unit tests pass (95.3%). Forces match Pure OpenCL to 5–7 decimals.
- Virial accumulation kernel returns NaN — chipStar bug. All 29 failures trace here.
- Pure OpenCL on same Arc A770: 30.84 Matom-step/s. chipStar HIP: 8.23 / 7.63 — ≈4× slower.

Case Study: GAMESS-GPU-HF

GAMESS-GPU-HF: 20,000 lines of HIP, ported.

HARTREE-FOCK ON AURORA PVC THROUGH chipStar

The application

GAMESS is a quantum chemistry suite (Fortran 77/90 + C/C++). The HF GPU library — 20,000+ lines of CUDA, ported to HIP — implements two-electron integral construction, matrix contractions, and eigensolves.

- HF method: starting point for most quantum chemistry.
- Original CUDA scaled to 4096 nodes on Summit (V100).
- Linear algebra via hipBLAS / hipSOLVER; FFTs via hipFFT.
- Verified correct on PVC. Porting effort: low.

What chipStar had to provide

Library shims to route hipBLAS / hipSOLVER / hipFFT through Intel oneMKL on PVC:

- H4I-HipBLAS / HipSOLVER / HipFFT
- One real spec gap surfaced: cl_ext_alive_only_barrier.

CASE STUDY · GAMESS-GPU-HF

Competitive with native CUDA on real HPC.

Hartree-Fock energy, 150-water cluster, STO-3G basis. 10 runs averaged. Same HIP source, three platforms.

Platform	Stack	SCF time (s)	vs A100	Notes
NVIDIA A100	CUDA 12.2.2	**1.66**		
Intel PVC	chipStar	2.84		
AMD MI250 (1 GCD)	ROCm 6.3.0	4.71		
── hardware peak ratios ──				**(upper bounds, A100 = 1.0)**
PVC mem-bound ceiling	1.3 TB/s		1.30×	vs A100's 1.0 TB/s
PVC compute ceiling	9.4 TF		0.56×	vs A100's 17.0 TFlop/s
			—	

1.7× off CUDA on real HPC code. *Within the 0.56–1.30× envelope set by hardware peaks. The portability layer is not the bottleneck.*

SECTION

AI Workloads

FUTURE WORK · AI WORKLOADS

Training: no. Inference: maybe.

HYPE TRAIN

Training — not today.

cuDNN and MIOpen are huge libraries — tens of thousands of lines of low-level vendor-specific intrinsics for tensor cores, MFMA, mma.

- The intrinsics are tuned to specific microarchitectures.
- A SPIR-V port would be portable, but not performant.
- Need vendor support.

Inference — maybe.

More tractable. Inference doesn't need vendor-tuned BLAS — proven by llama.cpp.

- llama.cpp already ships Metal, Vulkan, OpenCL, CUDA, ROCm, SYCL backends.
- chipStar can be more performant than SYCL
- The HIP path could be a competitive llama.cpp backend through chipStar.

SECTION

Getting Started

TRY IT

Three ways to get started.

1.

Docker (fastest)

```
docker pull chipspv/chipstar
```

Prebuilt containers. Fastest path to 'it just runs.'

2.

Full build with libraries

```
./scripts/configure_llvm.sh
./install_chipstar.py --all
```

Interactive TUI installer. Picks up library components.

3.

From source

```
cmake ../
```

Patched LLVM/Clang fork available; see docs/Getting_Started.md.

github.com/CHIP-SPV/chipStar
pvelesko@pglc.io

Q & A