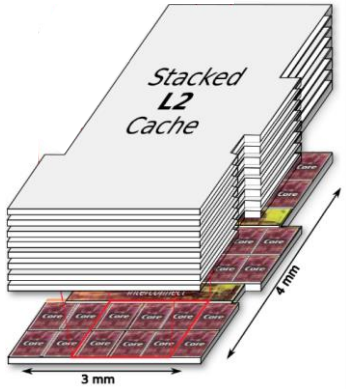


LARC: A Case Study in Enhancing CPUs with Copious 3D-Stacked Cache

NHR PerfLab Seminar 06. Sept. 2022

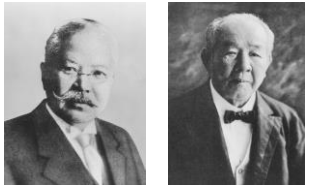
Jens Domke Dr. rer. nat. < jens.domke@riken.jp >
Supercomputing Performance Research Team, RIKEN R-CCS, Kobe, JP

An introduction to RIKEN

RIKEN Overview

President
Dr. Hiroshi Matsumoto

- **Founded 1917, Japan's first comprehensive research institute on natural sciences and engineering**
  - > Founding fathers: Dr. Jokichi Takamine, the samurai chemist, and Mr. Eiichi Shibusawa
- **Strategic research and development of the state-of the-art research infrastructure, in line with government's STI policy**
- **Top research quality and most internationalized among Japanese universities and National R&D Agencies**
- **Strong collaboration across laboratories and research centers, emphasis on interdisciplinarity**
- **Workforce: 3,000 research staff and 500 administrators**
- **Campus: 10 in Japan**
- **Annual budget: 900 M USD**
- **Subsidiary: RIKEN Innovation (founded in 2019, fully owned by RIKEN)**



Dr. Jokichi Takamine and Mr. Eiichi Shibusawa

RIKEN National Science Institute is Japan's most comprehensive research institute for the natural sciences, conducting cutting-edge research in a wide range of scientific fields.

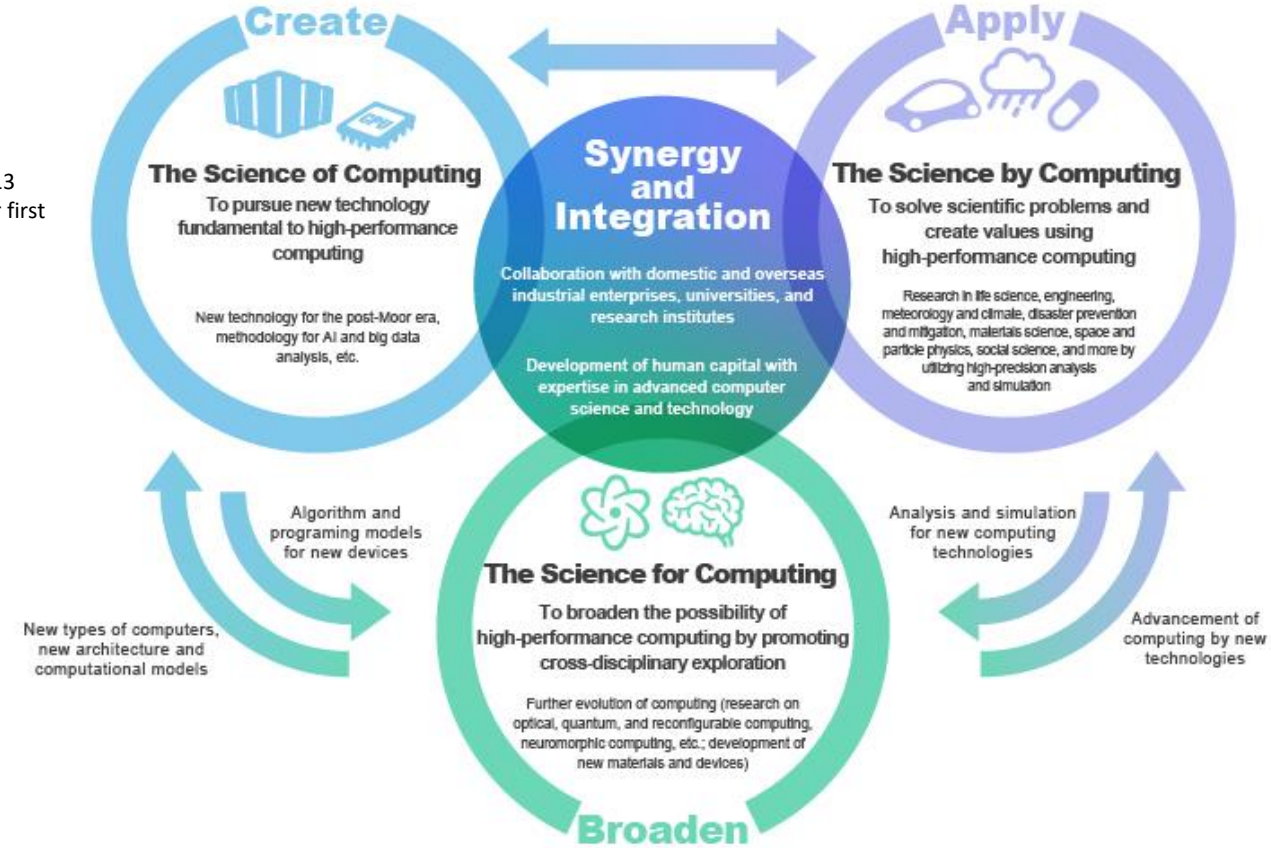
- Physics
- Developmental Biology
- Chemistry
- Space Science
- Health medicine
- Genetics
- Nanoscience
- Sustainable Resources
- Artificial Intelligence
- Computational Science
- Brain Science
- Energy
- Nuclear Physics
- Engineering
- Plant Science

- 1917 - RIKEN is founded.
- 1948 - RIKEN becomes KAKEN Scientific Research Institute Ltd.
- 1958 - RIKEN becomes a public corporation.
- 2003 - RIKEN becomes an Independent Administrative Institution.
- 2016 - RIKEN is designated as one of three National Research and Development Institutes in Japan.
- 2017 - RIKEN celebrates its 100th anniversary.

“The Science of Computing, by Computing, and for Computing”

Striving for excellence in science and becoming the cornerstone of Society 5.0

A research center out of 13 centers in RIKEN. The tier first national HPC center.



RIKEN R-CCS

Computer Science

Deputy Director
M. Sato

Programming Environment
M. Sato

Next Gen High Performance Architecture
M. Kondo

Advanced Processor Architectures
K. Sano

High Performance Big Data Systems
K. Sato

Parallel Numerical Technology
T. Imamura

From April 2022

High Performance AI Systems
Mohamed WAHIB

New Teams JFY2022 (plan)

(From July)
Supercomputing Performance Research

(New Team)
S5・Digital twin

Computational Science

Director
S. Matsuoka

Field Theory
Y. Aoki

Discrete Event Simulation
N. Ito

Molecular Science
T. Nakajima

Quantum Physics
S. Yunoki

Deputy Director
K. Nakajima

Biophysics
Y. Sugita

Climate Science
H. Tomita

HPC Engineering Applications
M. Tsubokura

Data Assimilation
T. Miyoshi

Structural Biology
F. Tama

Disaster Mitigation & Reduction
S. Oishi

HPC-and AI-driven Drug Development Platform Division

Biomedical Computational IntelligencebUnit
Yasushi Okuno

Medicinal Chemistry Applied AI Unit
Teruki Honma

Molecular Design Computational Intelligence Unit
Mitsunori Ikeguchi

AI-driven Drug Discovery Collaborative Unit
Yasushi Okuno

Office of the Fugaku Society 5.0 initiative

Director
S. Matsuoka

Deputy Director
M. Shinano

Operations and Computer Technologies

Facility Operations & Development
T. Tsukamoto

System Operations & Development
A. Uno

Software Development Technology Unit
H. Murai

HPC Usability Development
F. Shoji

Advanced Operation Technologies
K. Yamamoto

R-CCS Research Previous Highlights（JFY2021）

Achieved world's first five titles

(4 consecutive terms + new ML Perf HPC 1st place)

In four HPC performance rankings (Top500, HPCG, HPL-AI, Graph500), Fugaku won four titles consecutively from June 2020. In November 2021, also awarded first place in ML Perf HPC, a new overall performance evaluation of AI processing.

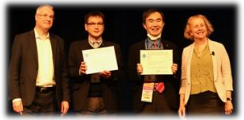
Fugaku's high overall performance in a wide range of fields, as well as its ability to make a significant contribution to the realization of Society 5.0/SDGs.

Gordon Bell Special Prize

Fight against COVID-19

Successfully developed a detailed and quantitative COVID-19 droplet and aerosol dispersion model using "Fugaku" for the first time in the digital transformation of infectious disease epidemiology. **Visualizing arised awareness of the importance of understanding droplet and aerosol infection changing behaviour not only in Japan, but also around the world.**

ITU-AJ Special Achievement Award



Data Assimilation Research

(Prediction of Sudden downpours, COVID-19 infection)

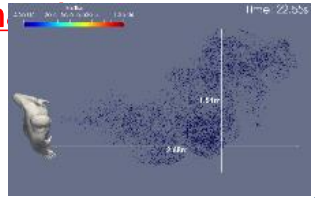
Using big data from weather radar, **a real-time of ultra-fast precipitation forecasting, was conducted in the Tokyo area using Fugaku during the Tokyo Olympic and Paralympic Games.**

Data assimilation methods developed in numerical weather forecasting were **applied to the forecasting of COVID-19 infections**

"GENESIS" new version released

Molecular dynamics (MD) simulations

A new version of GENESIS, optimized for 'Fugaku' by co-design, **more than 125 times faster** and with many new features, has been released as free software in 2020. **Work on the dyn structure of spike proteins on the surface of COVID-19 has analized successfully. RIKEN EIHO Award (RIKEN Significant Achievement Award)**

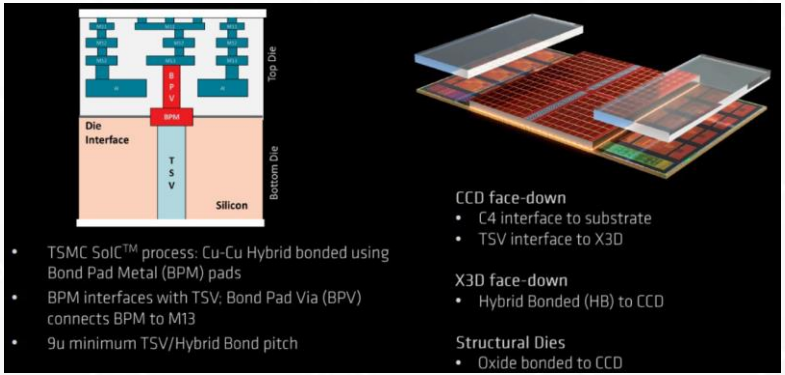


Outline

- Motivation – CPUs Empowered with High-capacity Cache
- From A64FX to a hypothetical LARC processor
- Evaluation Strategies and Results
  - Relevant HPC (Proxy-)Apps and Benchmarks
  - Simulating Unrestricted Locality with MCA
  - Cycle-level Accuracy: CPUs Simulated in gem5
- Discussions and Outlook towards 2028^{++}
- Summary

CPUs Empowered with High-capacity Cache

- Towards the future of post-Moore era: **quantum-, neuromorphic-, or reconfigurable computing** might be viable, but…
- 3D integrated circuit (IC) stacking can help classic von-Neumann CPUs now?



T. Morgan ""Milan-X" 3D Vertical Cache Yields Epyc HPC Bang For The Buck Boost" (The Next Platform)

Feasible?

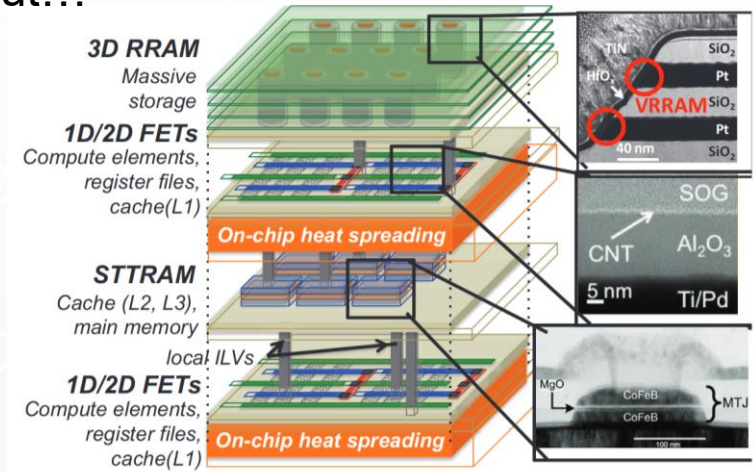


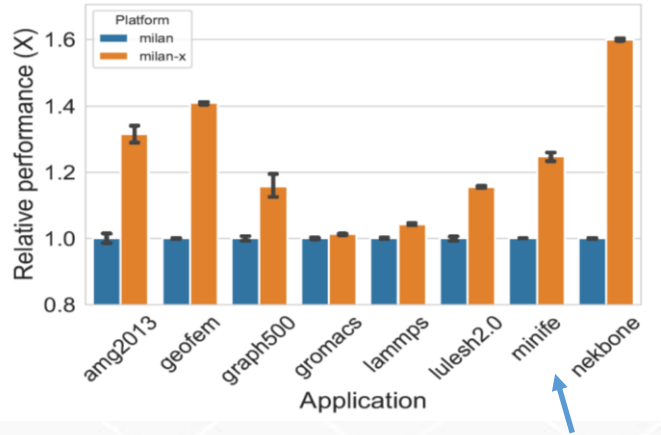
Figure 1. Monolithically integrated computing platform with CNFET-based logic circuits (for computing elements and memory access), STTRAM-based caches and main memory, and RRAM-based massive storage. The right-half portion of the figure includes the transmission electron microscopy (TEM) images of the different technologies. TEMs (top-to-bottom) from [Wong12], [Wei13], and [Smullen11].

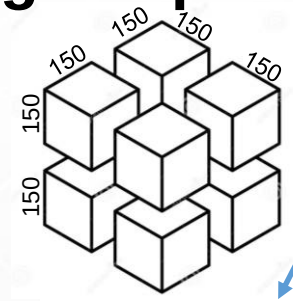
M. Shulaker et al. "Monolithic 3D Integration: A Path from Concept to Reality" in DATE'15

➔ Multiple discrete dies (comp/mem/IO) stacked

- ➔ Connected using coarse through-silicon vias (TSV), or
- ➔ Growing the 3D integrated circuit monolithically on the wafer

CPUs Empowered with High-capacity Cache





Peak ‘sweet spot’ around 150x150x150 w/ ~3x performance gain; Workload confinement to L3

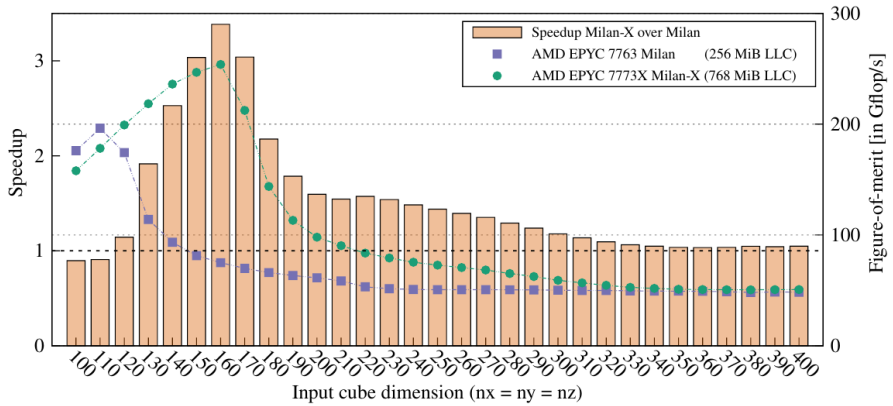


Figure 1. MiniFE: relative performance improvement of AMD EPYC 7773X Milan-X over AMD EPYC 7763 Milan, and Figure of Merit; Input problem scaled from 100×100×100 to 400×400×400; Both systems in dual-socket configuration; Benchmark run with 16 MPI ranks and 8 OpenMP threads

- Performance gain over 300^3
  - 3x by confining to enlarged L3
  - 8x by core parallelism w/ scaling => **total 24x speedup** **(**proxy for FugakuNEXT CPU)
  - *Caveat: assuming algorithmic strong scaling and process/packaging scaling*

CPUs Empowered with High-capacity Cache

From our previous research, we know:

- Majority of HPC is FP64-based and bandwidth-bound → *J. Domke et al. "Double-precision FPUs in High-Performance Computing: an Embarrassment of Riches?" in IPDPS'19*
- Matrix engines will not yield much performance in HPC → *J. Domke et al. "Matrix Engines for High Performance Computing: A Paragon of Performance or Grasping at Straws?" in IPDPS'21*

➔ Can we allocate silicon to other areas for performance gain?

Research question for this work:

- **Q1**: How many SRAM layers (how large L2) by 2028?
- **Q2**: Will HPC apps gain any speedup from it?
- **Q3**: Can we drive rapid "What-if" exploration with current simulation approaches?
- **Q4**: What is the right CPU arch. for future systems?
- **Q5**: Will apps / software stack need to adapt to large LLC?

} *HPC co-design*

From A64FX to hypothetical LARC Processor

- **A64FX** – the brain of Fugaku
  - Leading HPCG and Graph500
- **Cores:** 7 nm tech; **Arm** core with **SVE** for 512 bit vectors; 64 KiB L1i and L1d; **70.4 Gflop/s FP64**, support for FP16
- **CMG: 8 MiB L2**; 16-way set asso.; 256 B cache line; L1-L2 bus at 128 B (read) & 64 B (write); 1x **8 GiB HBM2**; **≈48 mm²** floorplan excl. I/O, etc.
- **CPU: 4 CMG**; 52 Arm cores (12+1 per CMG for user/OS); 4x HBM2 for **~1 TB/s** stream BW; **~120 W** incl. TOFU-D NIC; **32 MiB LLC** w/ L2 slices connected by a crossbar switch; 3.4 Tflop/s FP64; **≈400 mm²** floorplan

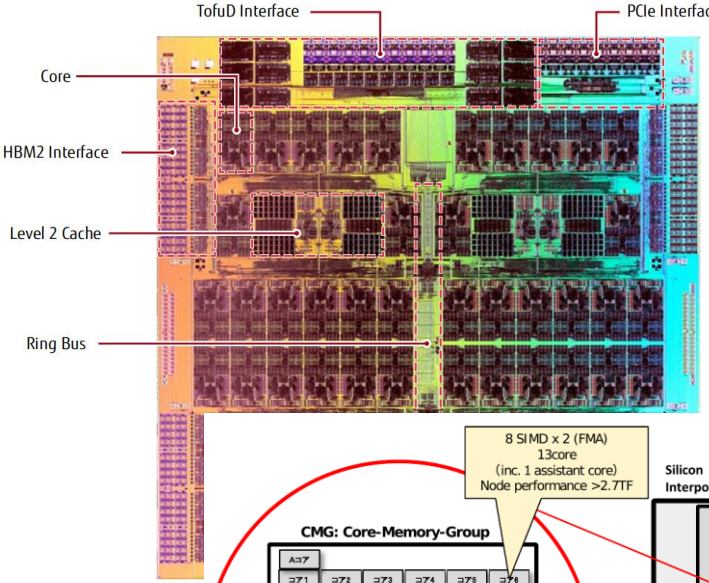
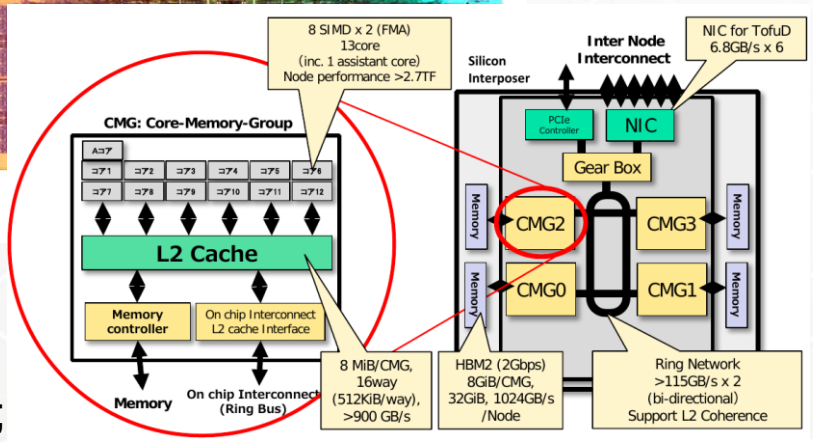


Figure 1
Die photo of A64FX CPU.

R. Okazaki et al. "Supercomputer Fugaku CPU A64FX Realizing High Performance, High-Density Packaging, and Low Power Consumption"



Y. Kodama et al. "Evaluation of the RIKEN Post-K Processor Simulator"

From A64FX to hypothetical LARC Processor

Computer simulations create the future

Projection towards 2028 with guesstimate and public roadmaps

- 1.5 nm in IEEE IRDS roadmap ➔ reduce silicon footprint ≈8x (≈2x per gen.) ➔ A64FX CMG at ≈6 mm^2 of silicon area ➔ replace L2 cache + controller with 3 cores ➔ double core count ➔ 32-core LARC CMG at ≈12 mm^2
- Stack L2 cache on top/below CMG (*derived from Shiba et al. "A 96-MB 3D-Stacked SRAM Using Inductive Coupling With 0.4-V Transmitter, Termination Scheme and 12:1 SerDes in 40-Nm CMOS" in IEEE TCAS-I*)
- **8x SRAM** dies connected with ThruChip Interface (**TCI**); capacity/bandwidth as function of #channels (N_{ch}), per-channel capacity (N_{cap} in KiB) and width (W in bytes), #stacked dies (N_{dies} =8), and operating freq. (f_{clk} in GHz)
- Scaling Shiba's 10 nm work results to 1.5 nm at ≈12 mm^2: N_{ch} ≈= 384, N_{cap} = 128 KiB, N_{dies} = 8 ➔ $N_{dies} \cdot N_{ch} \cdot N_{cap}$ **= capacity of 384 MiB** and f_{clk} = 1 GHz, W = 4 B ➔ $N_{ch} \cdot f_{clk} \cdot W$ = **bandwidth of 1536 GB/s** and read/write-latency of this SRAM cache is 3 cycles

From A64FX to hypothetical LARC Processor

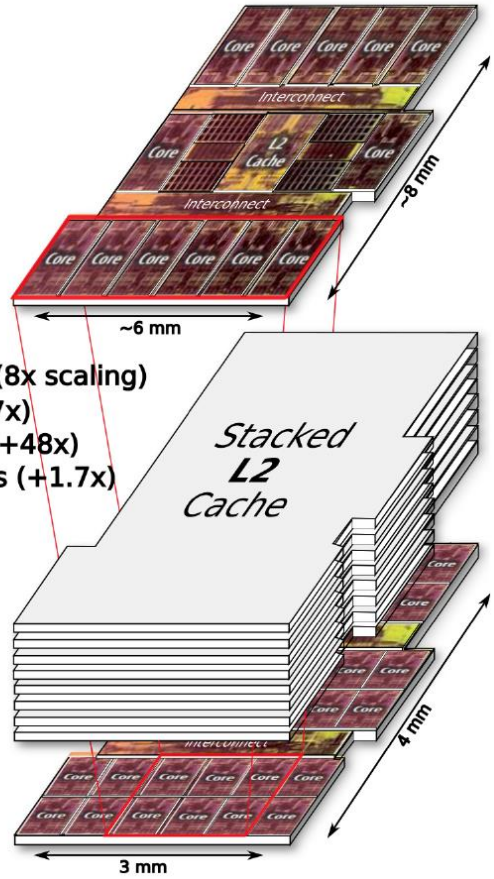
- New **LARC CMG** in 2028 timeframe
  - 32 A64FX-like cores w/ 64 KiB L1i and 64 KiB L1d, total of ≈2.3 Tflop/s
  - 384 MiB L2 with eight SRAM layers
  - (keep HBM2 to isolate perf. gains)
- New/**hypothetical LARC** CPU
  - die size similar to A64FX
  - 512 processing cores and **6 GiB of stacked L2 cache** with peak L2 bandwidth **of 24.6 TB/s**
  - peak HBM2 bandwidth of 4.1 TB/s
  - total **≈36 Tflop/s** in IEEE-754 FP64

A64FX CMG @7nm

CMG Area:	48 mm²
# Cores:	12
L2 Cache:	8 MiB
L2 B/W:	900 GB/s
HBM B/W:	256 GB/s

LARC CMG @1.5nm

CMG Area:	12 mm² (8x scaling)
# Cores:	32 (+2.67x)
L2 Cache:	384 MiB (+48x)
L2 B/W:	1536 GB/s (+1.7x)
# Dies:	8+1
# TCI Chan./Die:	384
# TCI Channels:	3072
TCI Channel Cap.:	128 KiB
HBM B/W:	256 GB/s



A64FX vs. LARC Core Memory Group Layout Comparison

Projecting LARC's Performance Improvement

- Can we look at a broad spectrum of HPC applications?

  ➔ Focus on real apps (w/ appropriate inputs), not benchmarks

- Which simulators can/should we use, and how long will it take?

  ➔ need a first-order approximation of a very large/fast cache

  ➔ gauge upper bound on perf. when all the memory-bottlenecks disappear

  ➔ Novel MCA-base 'infinite-L1' simulation

- Move to highly-detailed/slow simulators only if 1st projection exciting!

  ➔ gem5-base LARC simulation

127 Relevant HPC Proxy-Apps and Benchmarks

Computer simulations create the future

ECP	Workload
AMG	Algebraic multigrid solver for unstructured grids
CoMD	Generate atomic transition pathways bet. any 2 structures of protein
Laghos	Solves the Euler equation of compressible gas dynamics
MACSio	Scalable I/O Proxy Application
miniAMR	Proxy app for structured adaptive mesh refinement (3D stencil) kernels used by many scientific codes
miniFE	Proxy for unstructured implicit finite element or finite volume applications
miniTRI	Proxy for dense subgraph detection, characterizing graphs, and improving community detection
Nekbone	High order, incompressible Navier-Stokes solver based on spectral element method
SW4lite	Kernels for 3D seismic modeling in 4th order accuracy
SWFFT	Fast Fourier transforms (FFT) used in by Hardware Accelerated Cosmology Code (HACC)
XSBench	Kernel of the Monte Carlo neutronics app: OpenMC

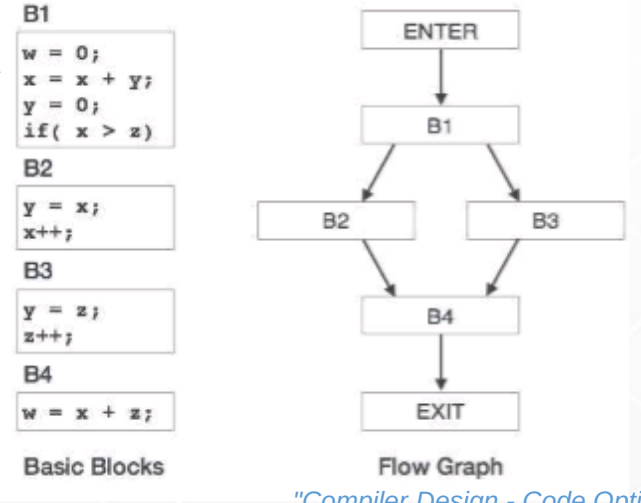
Bench	Workload
SPEC CPU	CPU[speed]/train / 20 test problems (10 int/single + 10 float/OMP)
SPEC OMP	train input / 14 OpenMP-parallelized HPC-focused benchmarks
PolyBench/C	EXTRALARGE / 30 single-threaded, scientific kernels (mem ∈ [16 KiB, 120 MiB])

Post-K	Workload
FFB	Unsteady incompressible Navier-Stokes solver by finite element method for thermal flow simulations
FFVC	Solves the 3D unsteady thermal flow of the incompressible fluid
MODYLAS	Molecular dynamics framework adopting the fast multipole method (FMM) for electrostatic interactions
mVMC	Variational Monte Carlo method applicable for a wide range of Hamiltonians for interacting fermion systems
NICAM	Benchmark of atmospheric general circulation model reproducing the unsteady baroclinic oscillation
NTChem	Kernel for molecular electronic structure calculation of standard quantum chemistry approaches
CCS QCD	Linear equation solver (sparse matrix) for lattice quantum chromodynamics (QCD) problem
RIKEN TAPP	scaled-down version of important kernels of above problems for quick gem5-based co-design

Bench	Workload
HPL	Solves dense system of linear equations Ax = b
HPCG	Conjugate gradient method on sparse matrix
Stream	Throughput measurements of memory subsystem
DLproxy	single-precision GEMM ops to approximate 2D deep CNN (224×224 ImageNet)
NPB OMP	class B / 10 proxy-apps of computational fluid dynamics (CFD) problems
NPB MPI	class B / 9 proxy-apps of computational fluid dynamics (CFD) problems

Simulating Unrestricted Locality with MCA

- **M**achine/Architecture **C**ode **A**nalyzer
- Recall **“Basic Blocks”** (BB):
  - straight-line code sequence
  - **no branches; 1 entry; 1 exit**
- Extracting BB is easy [for x64]
  - **Run application with Intel SDE**
  - Output contains: assembly, #executions, program counter, meta-data
  - but, **not 100% LLVM compat**ible

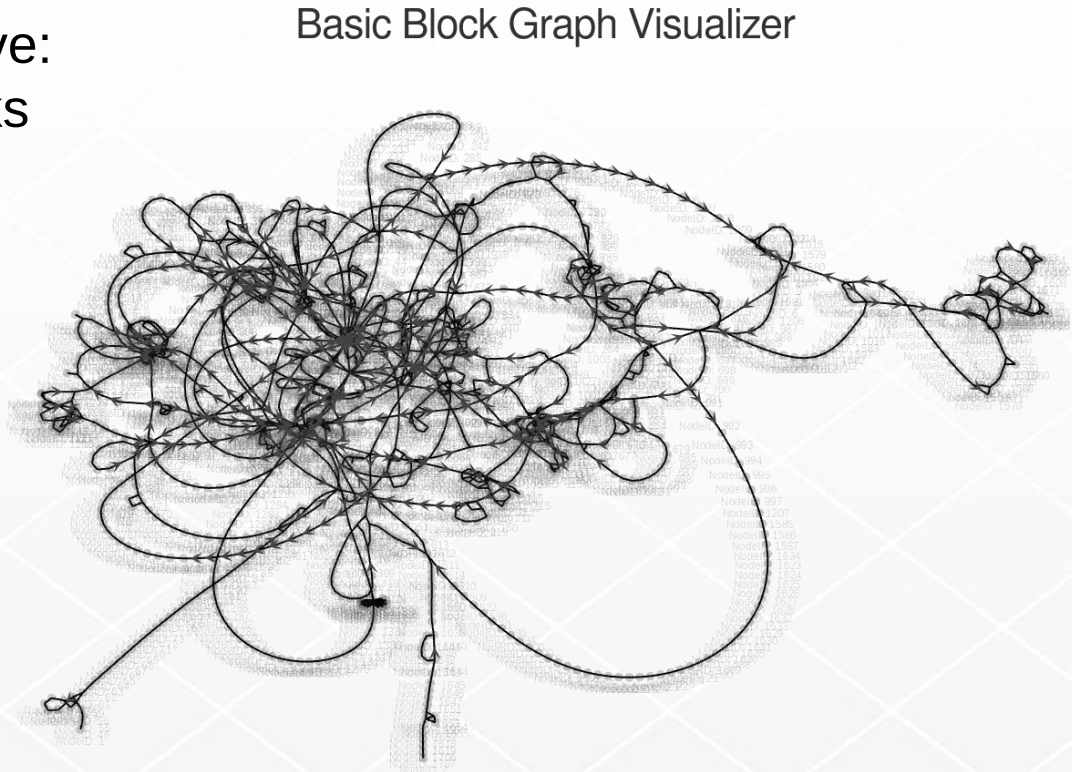


"Compiler Design - Code Optimization"
https://www.tutorialspoint.com/compiler_design/compiler_design_code_optimization.htm

```
BLOCK: 1295  PC: 481e13  ICOUNT: 4728  EXECUTIONS: 394  #BYTES: 78  FN: __intel_mic_avx512f_memset
XDIS 481e13: AVX512EVEX 62d17d48e703                  vmovntdq zmmword ptr [r11], zmm0
XDIS 481e19: AVX512EVEX 62d17d48e74301                vmovntdq zmmword ptr [r11+0x40], zmm0
XDIS 481e20: AVX512EVEX 62d17d48e74302                vmovntdq zmmword ptr [r11+0x80], zmm0
XDIS 481e27: AVX512EVEX 62d17d48e74303                vmovntdq zmmword ptr [r11+0xc0], zmm0
XDIS 481e2e: BASE     4d8d9b00020000                lea r11, ptr [r11+0x200]
XDIS 481e35: BASE     4881ea00020000                sub rdx, 0x200
XDIS 481e3c: AVX512EVEX 62d17d48e743fc                vmovntdq zmmword ptr [r11-0x100], zmm0
XDIS 481e43: AVX512EVEX 62d17d48e743fd                vmovntdq zmmword ptr [r11-0xc0], zmm0
XDIS 481e4a: AVX512EVEX 62d17d48e743fe                vmovntdq zmmword ptr [r11-0x80], zmm0
XDIS 481e51: AVX512EVEX 62d17d48e743ff                vmovntdq zmmword ptr [r11-0x40], zmm0
XDIS 481e58: BASE     4881fa00020000                cmp rdx, 0x200
XDIS 481e5f: BASE     7db2                          jnl 0x481e13
```

Simulating Unrestricted Locality with MCA

- Terminating applications have: START & END special blocks
- In-between **BB deps. can be represented as graph** eg. int main(){return 0;} ➔
- **Works for subsections or "kernels"** of the app as well (thanks to SDE)



➔ "Longest" (w.r.t time) path or Critical Path (CP) could be calculated

Simulating Unrestricted Locality with MCA

- **How** do we **know the "runtime"** for each BB in the graph?
- **LLVM-MCA** [for supported arch]
  - **IPC**, port pressure, CP length, and more statistics **for "any" ASM sequ.**
  - **WRN: first order estimate** and does not provide absolute perf. numbers
  - "optimistic" **load-to-use** latency (L1 hit)
- Similar tools (use all 4 for acc.):
  - **Intel IACA** (limited to x64; deprecated)
  - **OSACA** (RRZE-HPC)
  - **uiCA** (Saarland Univ)

J. Laukemann et al. "Automatic Throughput and Critical Path Analysis of x86 and ARM Assembly Kernels" in PMBS19

Throughput Analysis

```
* - Instruction not bound to a port

                                Port Pressure in cycles
|  0   |  1   |  2 - 2D  |  3 - 3D  |  4   |  5   |  6   |
-------------------------------------------------------------
|      |      | 0.5  0.5 | 0.5  0.5 |      |      |      |  .L22:
| 0.50 | 0.50 | 0.5  0.5 | 0.5  0.5 |      |      |      |  vmovapd 0(%r13,%rax),%ymm0
|      |      |          |          |      |      |      |  vfmadd213pd (%r14,%rax), \
|      |      |          |          |      |      |      |                %ymm1,%ymm0
|      |      |   0.5    |   0.5    | 1.0  |      |      |  vmovapd %ymm0,(%r12,%rax)
| 0.25 | 0.25 |          |          |      | 0.25 | 0.25 |  addq $32,%rax
| 0.25 | 0.25 |          |          |      | 0.25 | 0.25 |  cmpq %rax,%r15
|      |      |          |          |      |      |      |* jne .L22
| 1.00 | 1.00 | 1.5  1.0 | 1.5  1.0 | 1.00 | 0.50 | 0.50 |
```

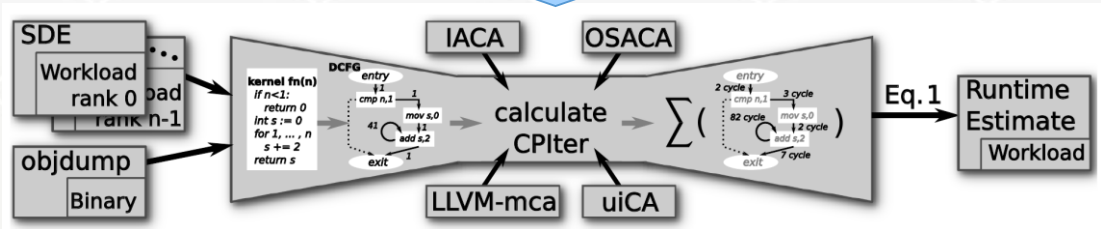
Critical Path Analysis

```
180 |  4.0 |  |  | vmovapd 0(%r13,%rax), %ymm0
181 |  4.0 |  |  | vfmadd213pd (%r14,%rax), %ymm1, %ymm0
182 |  5.0 |  |  | vmovapd %ymm0, (%r12,%rax)
    | 13.0 |
```

Loop-carried Dependencies Analysis

```
183 | 1.0 | addq     $32, %rax                    | [183]
```

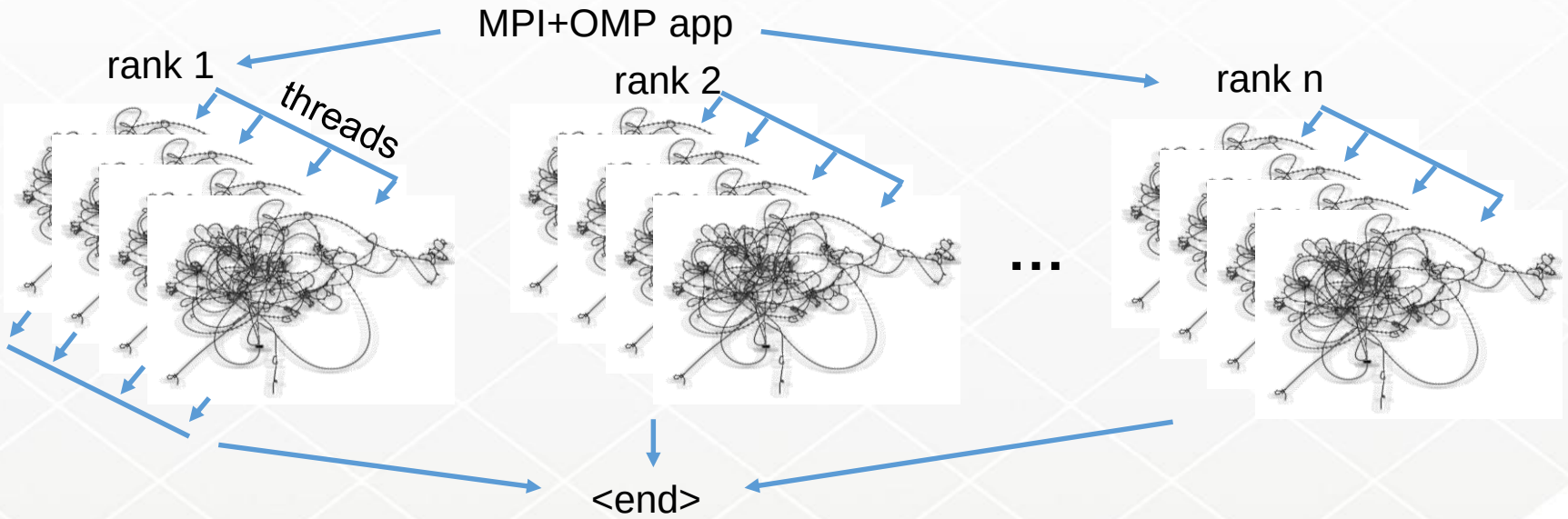
Feed our framework



Simulating Unrestricted Locality with MCA

- Estimate “easy” for fork-join model execution (1 thread/core)

➔ Theory: **App. Runtime** $t_{app} := \dfrac{\max\limits_{r \in \text{ranks}} \left(\max\limits_{t \in \text{threads}_r} \left(\sum\limits_{\text{edges } e \in \text{CFG}_{t,r}} \text{CPIter}_e \cdot \#\text{calls}_e \right) \right)}{\text{processor frequency in Hz}}$



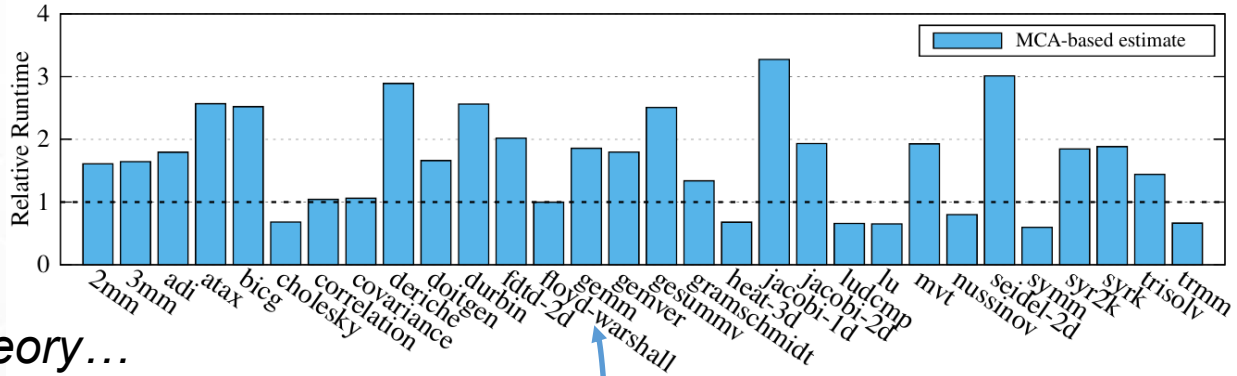
- Other execution models (async; many threads; e.g. JVM) more tricky ☹

Simulating Unrestricted Locality with MCA

Computer simulations create the future

Validation experiment 1

- Against PolyBench/C w/ MINI input (≅16 KiB)
  - ➔ should fit in L1D
  - ➔ MCA and real HW "should" match *in theory*…



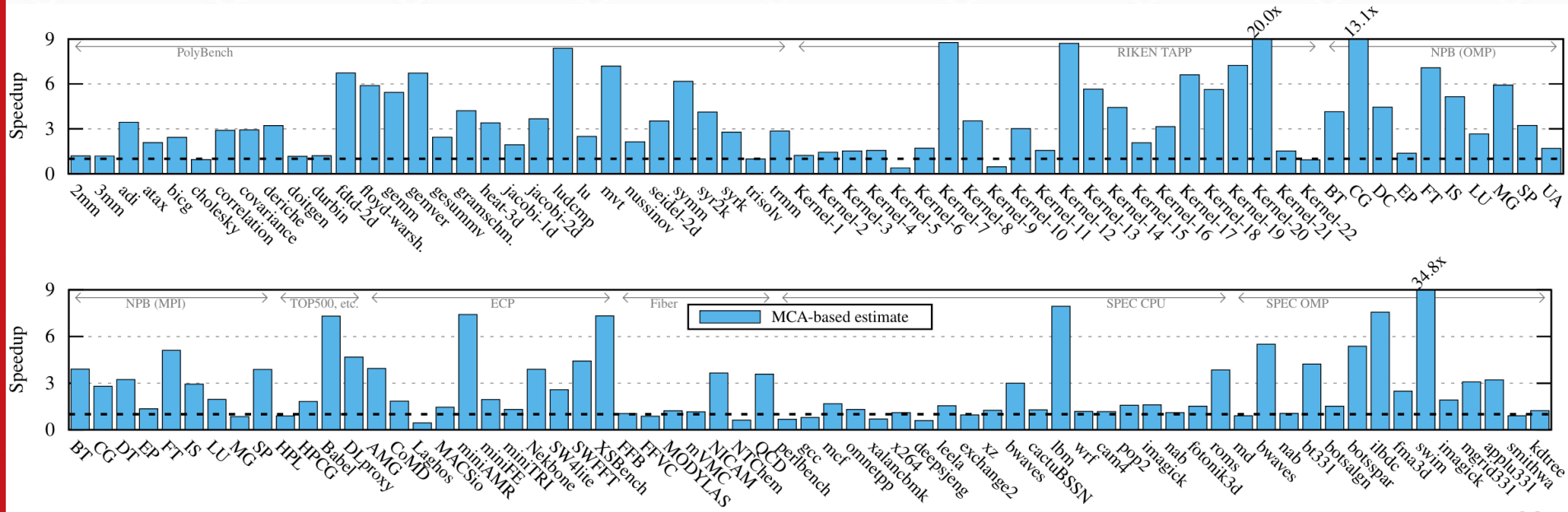
- For 73% of BMs, MCA sim. reasonably accurate: 2x slower-to-2x faster range (cf. other, much slower, simulators such as SST, gem5, etc.)

Validation experiment 2

- GEMM *should* be accurate, but is handwritten ➔ new test with MKL
- For input sizes MINI, …, EXTRALARGE: 6.4x, 75%, 11%, 1.9%, and 1.5% and 2 Gflop/s, …, 32 Gflop/s (i.e., close to per-core-peak)
  - ➔ not compute-bound for small inputs ➔ discrepancy betw. real HW and MCA

Simulating Unrestricted Locality with MCA

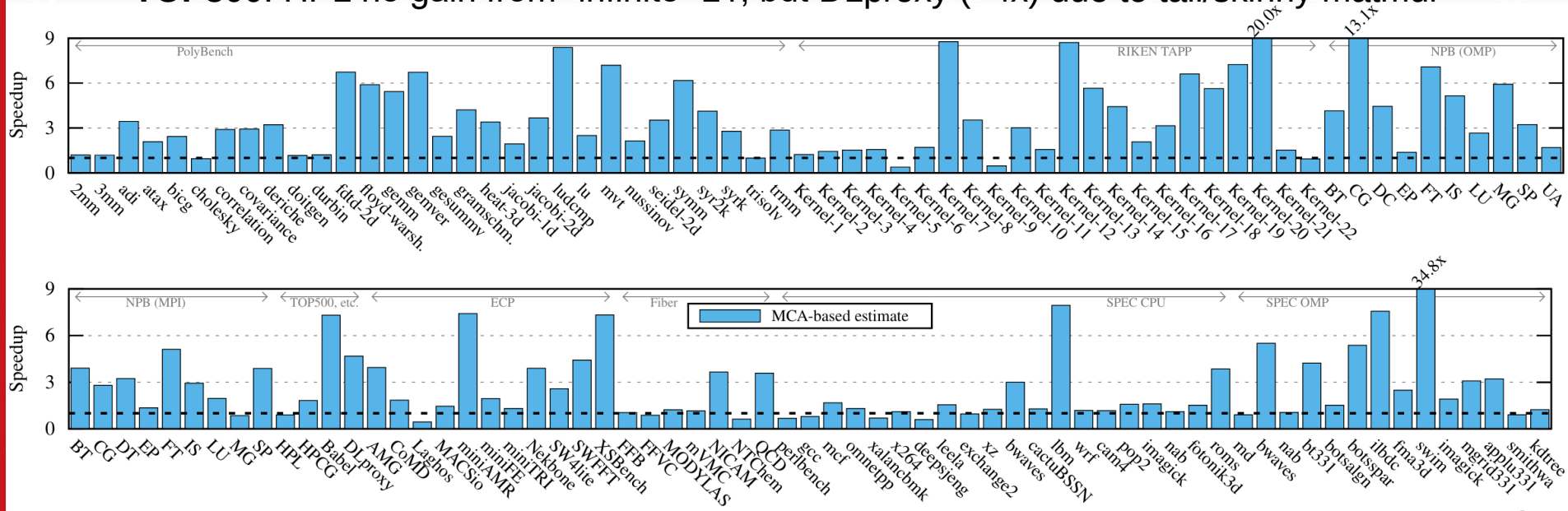
- 48-core, 2.2 Ghz dual-socket Broadwell E5-2650v4 ➔ baseline & get SDE data
- Initial **sweep of MPI/OpenMP** config. (strong-scale) for fastest time-to-solution
- Focus only on solver/kernel times (not init/post-processing), except for SPEC



Simulating Unrestricted Locality with MCA

Computer simulations create the future

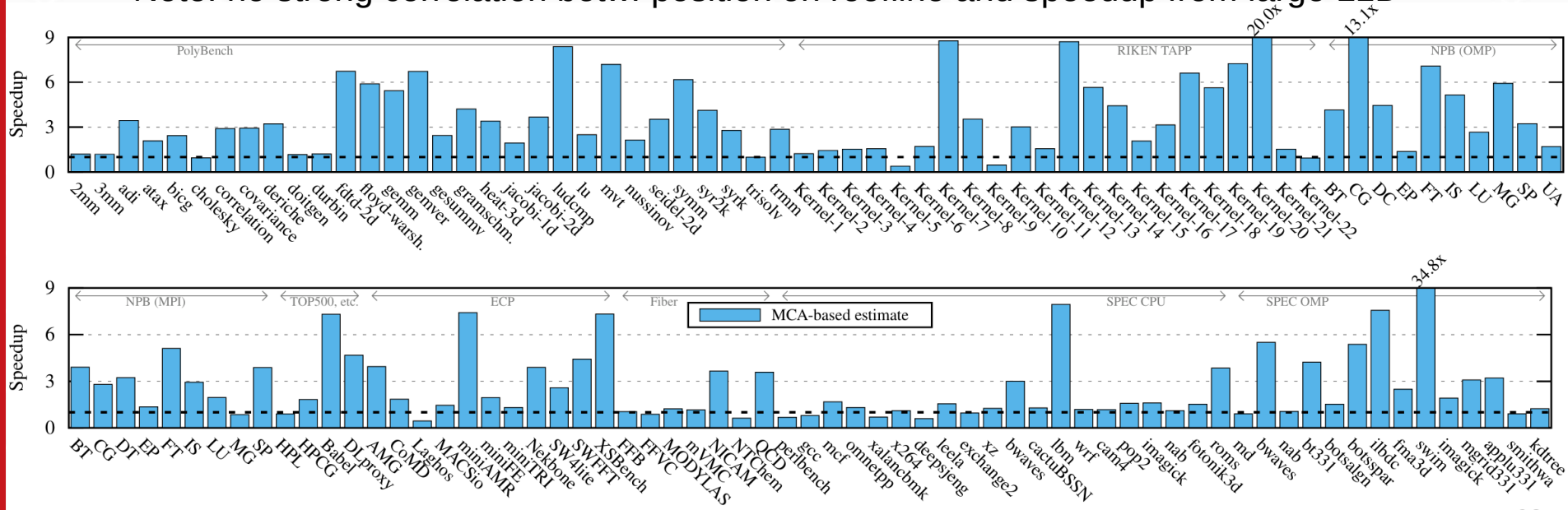
- **PolyBench: 8.4x** for ludcomp; compute-bound (2mm, etc) ➔ no benefit; ***GM*=2.9x**
- **TAPP: 20x** speedup (kernel of FFB); overall ***GM*=2.9x** speedup; odd: slowdown for K5/9
- **NPB: >13x** for conjugate gradient benchmark; total ***GM*=4x** (OMP) and ***GM*=2.3x** (MPI)
- **TOP500**: HPL no gain from “infinite” L1, but DLproxy (>4x) due to tall/skinny matmul



Simulating Unrestricted Locality with MCA

- **ECP**: **7.3x** for XSBench and **7.4x** for miniAMR; ***GM*=2.5x**; odd: slowdown for Laghos
- **Fiber**: **3.7x** for NICAM and **3.6x** for QCD; ***GM*=1.4x**
- **SPEC CPU[speed]**: ***GM*=1.0x (Int)** and ***GM*=1.9x (Float)** and **SPEC OMP**: ***GM*=2.9x**
- Note: no strong correlation betw. position on roofline and speedup from large L1D

J. Domke et al. "Double-precision FPUs …" in IPDPS'19



Cycle-level Accuracy: CPUs Simulated in gem5

- Employ open-source system architecture simulator **gem5**
  - **Supports Arm**, x86, and RISC-V CPUs, and GPUs
  - Extendable with memory models and plugins for higher fidelity (e.g. RUBY)
  - Use "**syscall emulation**" to execute applications without booting Linux ("FS" mode)
- RIKEN **forked gem5 for A64FX co-design** (github.com/RIKEN-RCCS/riken_simulator)
  - Lacked of support for dynamically linked binaries
  - Lacked adequate memory management (ignores free() calls)
  - No support for more than 16 cores (issue in coherence protocol)

  } *FIXED*

  - No multi-rank MPI-based programs ☹ ➔ **MPI stub library and 1-rank sims**
  - Cannot simulate more than one A64FX CMG ☹ ➔ if we assume weak-scaling HPC codes across NUMA/compute node domains ➔ **1 CMG is reasonable proxy**

Cycle-level Accuracy: CPUs Simulated in gem5

LARC' CMG not easily replicated

- Gem5 requires pow2 for L2 size

  ➔ 2 Configs.: **LARC$_c$**< LARC < **LARC^a**
- **Conservative** LARC: 256 MiB at 800 GiB/s
- **Aggressive** LARC: 512 MiB at 1.6 TiB/s
- Cycle-level accurate **simulation w/ gem5** (patches to fix bugs and scale cores)
- Base: **1x 2.2 Ghz CMG of Fugaku's A64FX**
- Additional: **A64FX^{32}** to separate effect of core increase from cache increase
- Use number of L2 banks to control BW
- Employ CRIU for Checkpoint/Restore

Table 1. Chip area and simulator configurations for gem5

	A64FX$_S$	A64FX^{32}	LARC$_C$	LARC^A
CORE CONFIG: Arm v8.2 + SVE, 512bit SIMD width, 2.2 GHz, OoO 128 ROB entries, dispatch width 4				
Cores	12	32	32	32
CMGs	4	4	16	16
BRANCH PREDICTOR: Bi-mode: 16K global predictor, 16K choice predictor				
PER-CORE L1D: 64KiB 4-way set-assoc, 3 cycles, adjacent line prefetcher				
L2 CONFIGURATION: 16-way set-associative, 37 cycles, inclusive, 256 B block				
L2 CACHE PER CMG:				
L2 size	8 MiB		256 MiB	512 MiB
BW	~ 800 GB/s		~ 800 GB/s	~ 1600 GB/s
L2 CACHE AGGREGATED:				
L2 size	32 MiB		4096 MiB	8192 MiB
BW	~ 3.2 TB/s		~ 12.8 TB/s	~ 25.6 TB/s
MAIN MEMORY: 32 GiB HBM2, 4 channels, 256 GB/s				

Cycle-level Accuracy: CPUs Simulated in gem5

Computer simulations create the future

Validation experiment 1

- Employ STREAM Triad benchmark
- Vector sizes of 128 KiB per core; scale cores

  ➔ Matches 1x A64FX CMG and close to target BW with 792 GB/s and 1450 GB/s

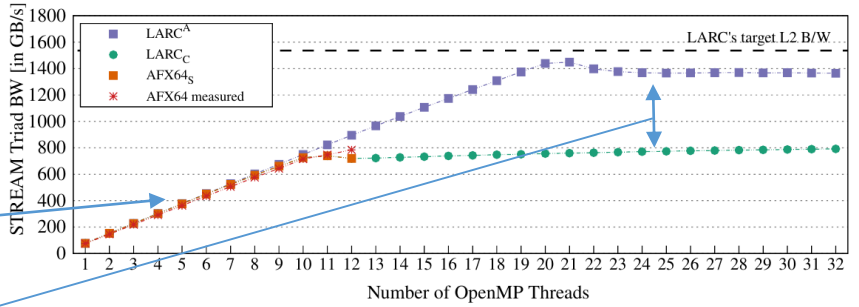


Figure 8. Validation of the simulated STREAM Triad bandwidth for fixed 128 KiB vectors per core; $A64FX_S$ scaled to 12 cores; Real A64FX measurements on 1 CMG for reference;

Validation experiment 2

- Fix number cores at 32 (and 12 for $A64FX_S$)
- Scale vector size from 2 KiB to $^1/_3$ GiB
- 2.7x higher core count ➔ 2.6x higher aggregated L1 bandwidth
- Vector sizes fit L2 cache ➔ similar to Exp.1
- Sizes beyond L2 ➔ expected HBM2 speed

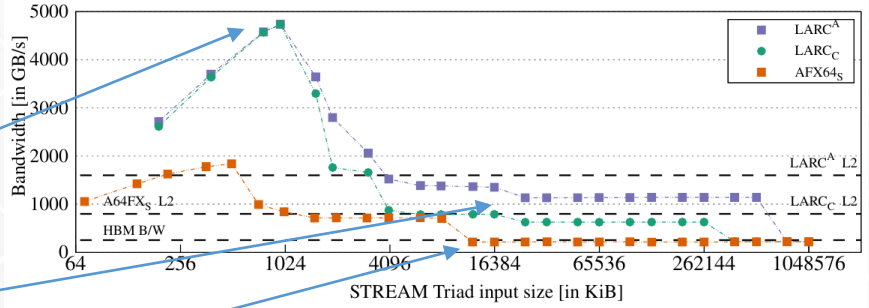
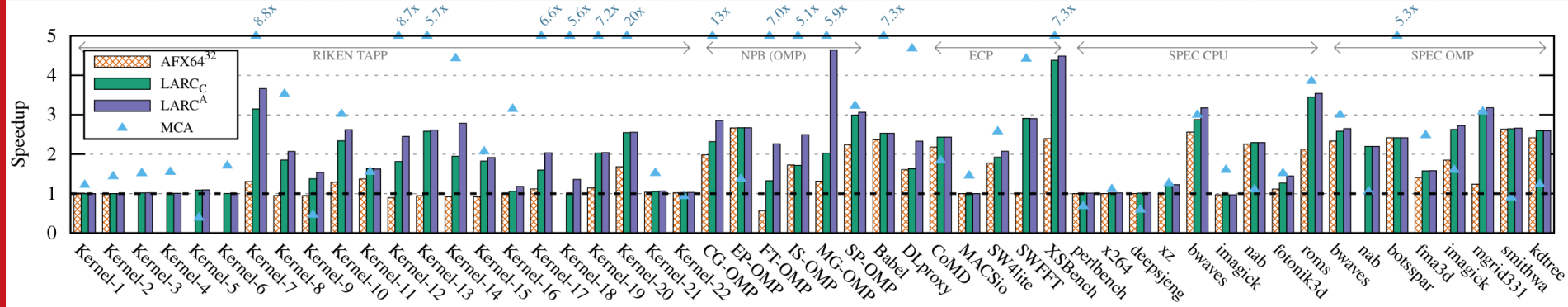


Figure 9. Validation of the simulated STREAM bandwidth for both LARC configurations with 32 cores (vs. $A64FX_S$ with 12 cores); STREAM Triad input range from few KiB to 1 GiB;

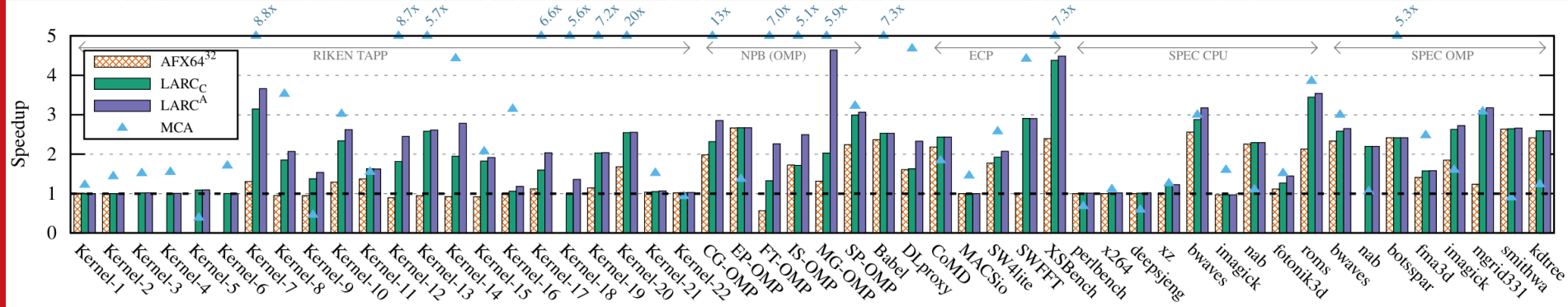
Cycle-level Accuracy: CPUs Simulated in gem5

- Baseline is A64FX$_S$ and speedup plotted for **A64FX^{32}, LARC$_C$, LARC^A**
- Results collected: 16-node cluster for **>6 month for 52 benchmarks**
  - Excluding some crashes and some much longer running BMs ; excluding MODYLAS, NICAM, and NTChem which need >1 ranks ; excluding MPI-only NPB ; excluding PolyBench (showing no noticeable benefit from cores or cache increase (only *GM*=4%)
  - Estimated avg. 10k-30k times slowdown per simulated core ☹
- Blue dots for MCA-based estimates for same BM (*caveat: multi-rank & x86*)
- Focus on time-to-solution for solver/kernel only (except SPEC)



Cycle-level Accuracy: CPUs Simulated in gem5

- **LARC$_c$**: avg. speedup of ≈1.9x and peak of ≈4.4x (see XSBench)
- **LARC^a**: avg. speedup of ≈2.1x and peak of ≈4.6x (see MG-OMP)
- **MG-OMP**: ≈1.3x from extra cores, ≈2x from 256 MiB L2, ≈4.6x from 512 MiB
- SPEC CPU Int: matches zero-speedup estimates from MCA
- Kernel 8, 9, 12–15, FT-OMP: slowdown from cache contention in A64FX[32]
- EP-OMP, CoMD, other compute-bound: expected benefit from more cores only



Cycle-level Accuracy: CPUs Simulated in gem5

Computer simulations create the future

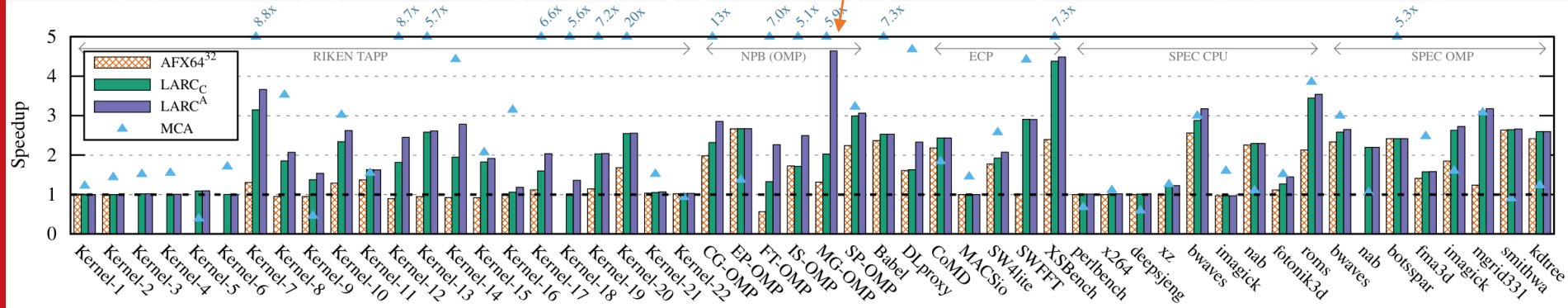
Reasons for speedup w/ A64FX^{32}

- App is compute-bound ➔ valid result
- Compute- and memory-bound in different sections ➔ valid
- Highly latency-bound ➔ speedup from larger aggregate L1 cache ➔ valid
- **Poor baseline** (e.g. BabelStream) ➔ **misleading**

➔ Reduction in cache-miss rate consistent with the performance gains

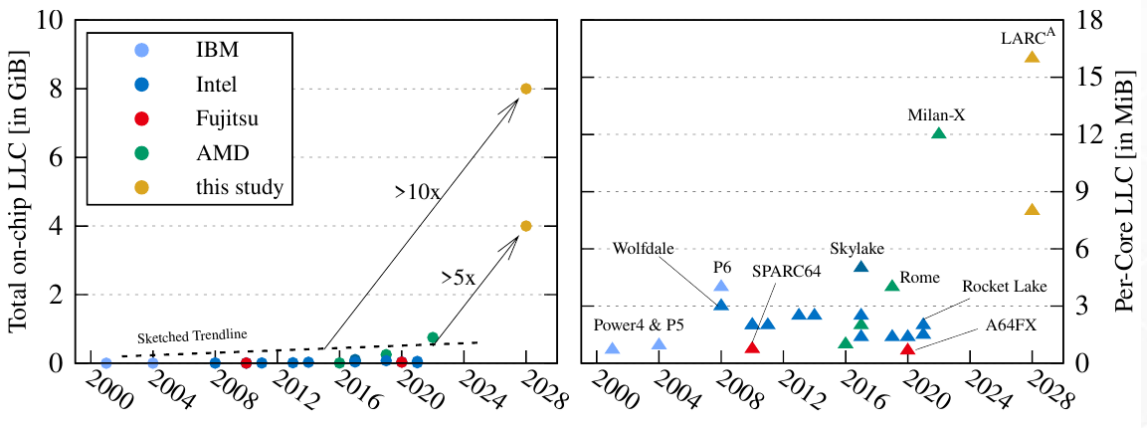
Table 2. L2 cache-miss rate [in %] of representative proxies

Proxy-App	A64FX$_S$	A64FX^{32}	LARC$_C$	LARC^A
Kernel 12	36.6	47.6	10.5	9.1
Kernel 17	46.7	49.5	48.7	34.8
Kernel 19	73.8	69.6	49.1	48.9
FT-OMP	11.6	48.2	6.4	3.8
MG-OMP	59.8	70.9	29.4	0.4
XSBench	32.1	36.4	0.1	0.1



Discussions and Outlook towards 2028++

- **31 of 52 apps show ≥2x speedup** on LARC[a] compared to baseline
- For >$^2/_3$ (24 of 31) the perf. gains come from 3D-stacked cache (>10% gain)
- Our study explores a **radical shift in on-chip LLC capacity**



- Assuming **ideal scaling** of apps and **same area** (i.e. 16x CMG LARC vs. 4x CMG A64FX) ➔ we gain **betw. 4.91x** (xz; SPEC) and **18.57x** (MG-OMP) **performance by 2028**
- Ideal scaling and same area: we gain ***GM* ≈10x speedup from LARC**
- **Application-specific restructuring** to utilize large caches increases benefit

H. Ltaief et al. "Meeting the Real-Time Challenges of Ground-Based Telescopes Using Low-Rank Matrix Computations" in SC '21

Discussions and Outlook towards 2028++

Computer simulations create the future

LARC power estimated at **around 547 W** (but likely less for BW-bound apps)

- A64FX's peak power ≈122 W (➔ 1.98 W /core and 3.75 W /memory-interface)
  ➔ LARC CMG in 7 nm ≈67.1 W ➔ power projection (IRDS): 27.4 W in 1.5 nm
  ➔ for 16x CMGs a total power of 438 W (excl. L2 cache)
- 4 MiB SRAM L2 in 7 nm at 64 mW of static power (~90-98% static, rest dyn.)
  ➔ at 384 MiB in 1.5 nm and (pessimistic) no power improvement: 6.14 W
  ➔ for 16x CMGs additional 98.3 W static power + 10.07 W dynamic (9:1)

LARC thermal considerations

- ~550W no issue (see Nvidia), but: power density (W/mm^2) and SRAM layers
- Our power estimates are pessimistic ➔ room for improvement
- Stack L2 layers below cores (or alternatives: we are working on it ☺)
- Direct-die cooling, high-κ thermal compound, microfluid cooling, thermal-aware floorplanning, task-scheduling and data-placement, etc. ➔ more research opportunities

Summary

- Exploring the benefit of large **3D-stacked SRAM for HPC** codes
- Envisioned **hypothetical, 512-core, 36 Tflop/s LARge Cache** (LARC) processor with 6 GiB of SRAM (L2 cache) at 24.6 TB/s for 2028 timeframe
- Developed **MCA-based performance prediction** framework ➔ assumes "infinite" L1D; orders of magnitude faster than gem5; reasonable accurate
- **Utilized gem5** to simulate A64FX in two variants and LARC in two variants
- Predicted LARC's and 3D-stacked SRAM peak power consumption
- **Explored performance gain for >120 HPC proxy apps** and benchmarks
- Assuming **ideal scaling** and **same area** ➔ we gain ***GM* ≈10x speedup from LARC** for our bandwidth-bound HPC applications
- Open-sourced **framework** (https://gitlab.com/domke/LARC) and **our results** (https://zenodo.org/record/6420659; which contain more valuable data)

Join RIKEN R-CCS or other RIKEN centers

Current SPR Team Members

Jens Domke

- **Position**: TL
- **Country**: Germany
- 2021 Researcher @R-CCS
- 2017 Postdoc @Tokyo Tech
  & 2019 Postdoc @R-CCS
- 2017 PhD in CS (TU Dresden)
  2010 Master in Mathematics
- **Research**: HPC networks, HPC performance analysis & modelling, co-design

Ivan R. Ivanov

- Master student @Tokyo Tech
- **Position**: Part-timer / Trainee
- **Country**: Bulgaria
- **Research**: HPC networks, GPU➔CPU transpilation

Team Assistant:
Maekawa, Chikako

Job & Collaboration Opportunities

Computer simulations create the future

- Collaborations and job opportunities:
  - **We are hiring!** Check out our research teams and open positions:
    https://www.riken.jp/en/research/labs/r-ccs/ and < jens.domke@riken.jp >
    https://bit.ly/3faax8v
    https://bit.ly/3tLVwBZ ← **Currently hiring for SPR Team!**
- Internship/fellowship for students (Bachelor→PhD):
  - Fellowship: https://www.riken.jp/en/careers/programs/index.html
  - Internship: https://www.r-ccs.riken.jp/en/about/careers/internship/
- Supercomputer **Fugaku**:
  - Apply for node-hours: https://www.r-ccs.riken.jp/en/fugaku/user-guide/
  - Interactive, virtual tour: https://www.r-ccs.riken.jp/en/fugaku/3d-models/ and https://www.youtube.com/watch?v=f3cx4PGDGmg

Supplementary material

Things you hear…

“Wanna do HPC? Then you need dense nodes (SuperPODs) and full-bisection bandwidth fat-trees.”

--NVIDIA/Mellanox

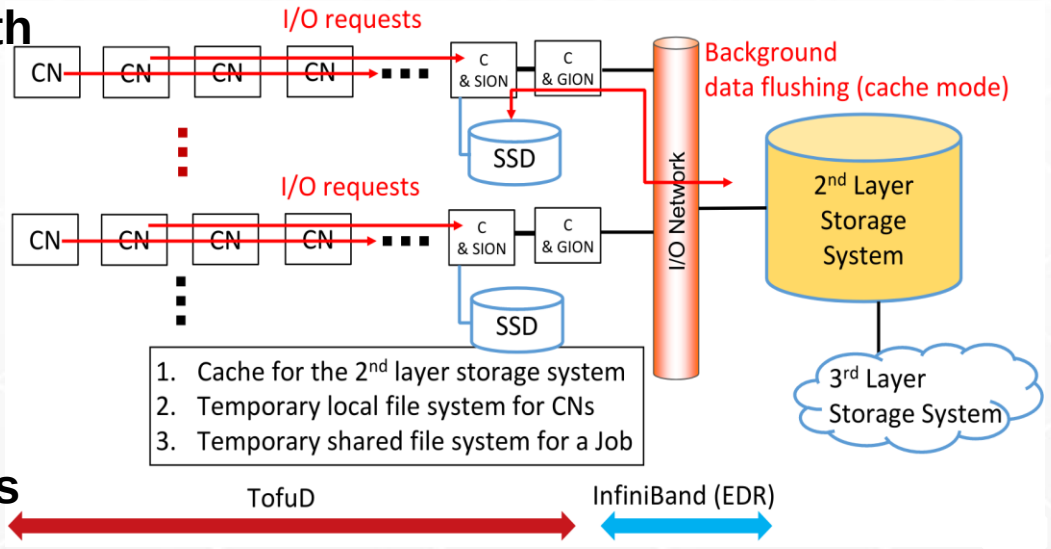
Challenges – Future Scale-out & Diversification

- No more gains from Moore's law

➔ Bigger HPC systems w/ more nodes (maybe island design for specialization)
➔ **Need for memory bandwidth**
➔ Larger interconn. networks

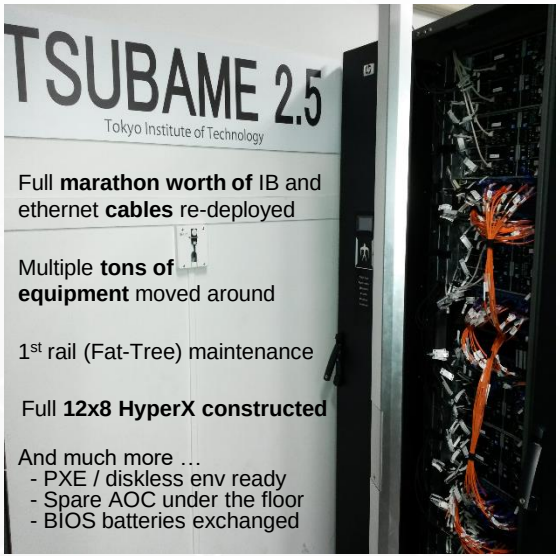
E.g.: Supercomputer Fugaku

- >158k compute nodes
- 3 networks / topologies
  - CN: 24x23x24 **TofuD** (w/ 2x3x2 subgr.) as **6D torus**
  - Storage: EDR **InfiniBand** for every 16th CN (with fat-tree? topology)
  - Management + outside world: **Ethernet**



Y. Tsujita "Status of Lustre-Based Filesystem at the Supercomputer Fugaku"

Opportunity for new topologies – HyperX



➔ **First large-scale 2.7 Pflop/s (DP) HyperX installation in the world!**

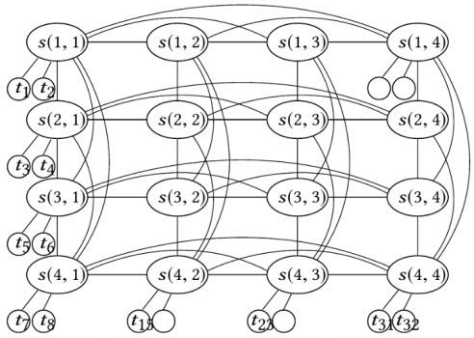


Fig.1: HyperX with n-dim. integer lattice (d_1,…,d_n) base structure fully connected in each dim.

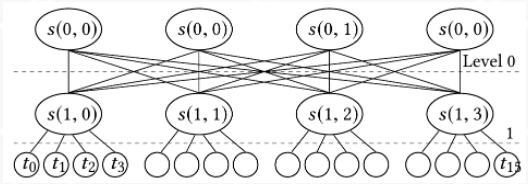
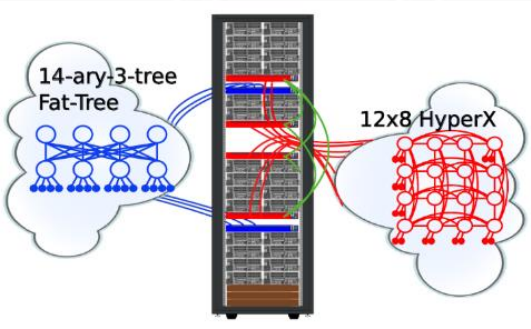


Fig.2: Indirect 2-level Fat-Tree

TokyTech's 2D HyperX:

- **24 racks** (of 42 T2 racks)
- 96 QDR switches (+ 1st rail) **without adaptive routing**
- 1536 IB cables (720 AOC)
- **672 compute nodes**
- **57% bisection bandwidth**



J. Domke "HyperX Topology: First at-scale Implementation and Comparison to the Fat-Tree"

Theoretical Advantages (over Fat-Tree)

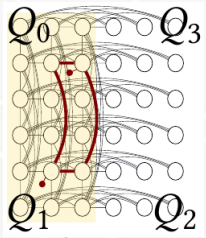
- **Reduced HW cost** (less AOC / SW)
- **Only needs 50% bisection BW**
- **Lower latency** (less hops)
- **Fits** rack-based **packaging**

Opportunity for new topologies – HyperX

- TSUBAME2's older gen. of **QDR IB** hardware has **no adaptive routing** ☹
- HyperX with **static/minimum routing** suffers from **limited path diversity** per dimension
  ➔ results in high congestion and low (effective) bisection BW

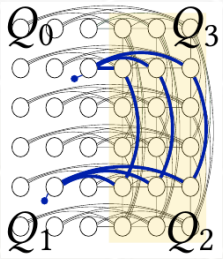
Mitigation Strategies???

- Option 1: Alternative Job Allocation
- **Option 2: Non-minimal, Pattern-aware Routing (PARX)**

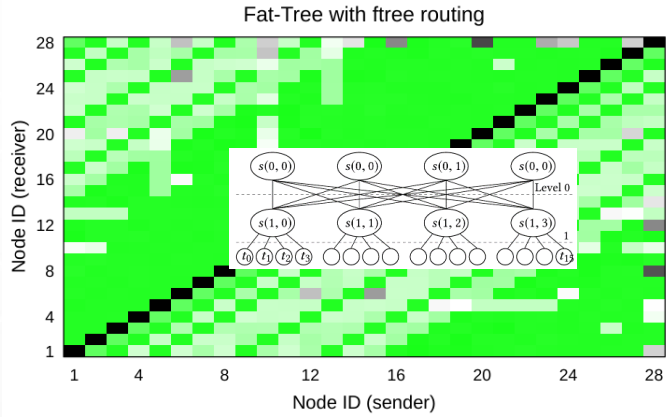


Minimum paths

➔ *combine* ➔

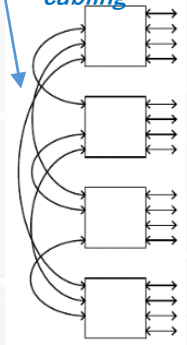


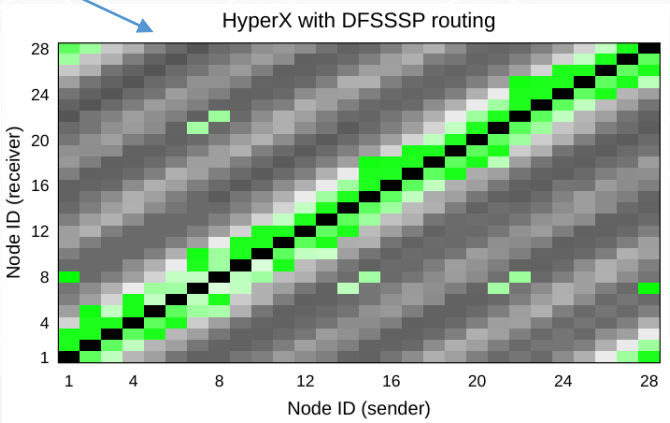
Forced detours



Measured BW in mpiGraph for 28 Nodes

HyperX intra-rack cabling





Opportunity for new topologies – HyperX

Computer simulations create the future

1:1 comparison (as fair as possible) of 672-node 3-level Fat-Tree and 12x8 2D HyperX

- NICs of 1st and 2nd rail even on same CPU socket
- Given our HW limitations (few "bad" links disabled)

Wide variety of benchmarks and configurations

- 3x Pure MPI benchmarks
- 9x HPC proxy-apps
- 3x Top500 benchmarks
- 4x routing algorithms (incl. PARX)
- 3x rank-2-node mappings
- 2x execution modes

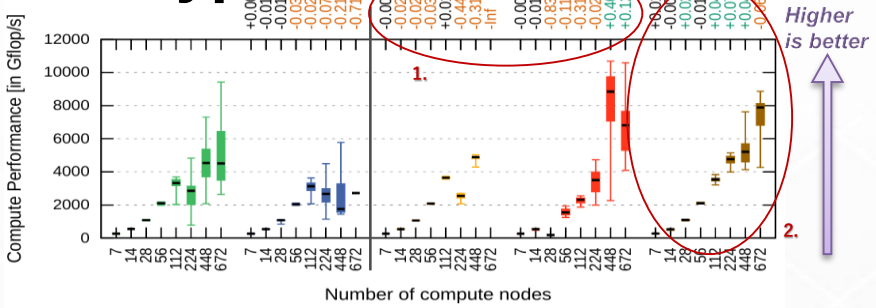


Fig.3: HPL (1GB pp, and 1ppn); scaled 7→672 cn

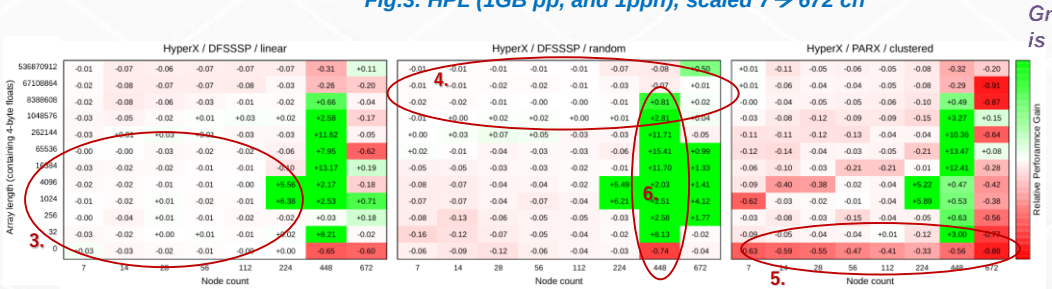


Fig.4: Baidu's (DeepBench) Allreduce (4-byte float) scaled 7→672 cn (vs. "Fat-tree / ftree / linear" baseline)

Primary research questions

Q1: Will reduced bisection BW (57% for HX vs. ≥100% for FT) impede performance?

Q2: Two mitigation strategies against lack of AR? (→ e.g. placement vs. "smart" routing)

1. Placement mitigation can alleviate bottleneck
2. HyperX w/ PARX routing outperforms FT in HPL
3. *Linear* good for small node counts/msg. size
4. *Random* good for DL-relevant msg. size (+/- 1%)
5. "Smart" routing suffered SW stack issues
6. FT + ftree had bad 448-node corner case

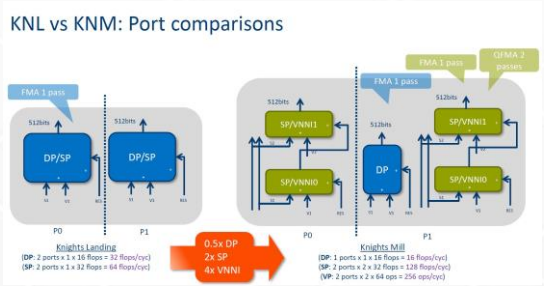
Conclusion

HyperX topology is promising and cheaper alternative to Fat-Trees (even w/o adaptive *R*) !

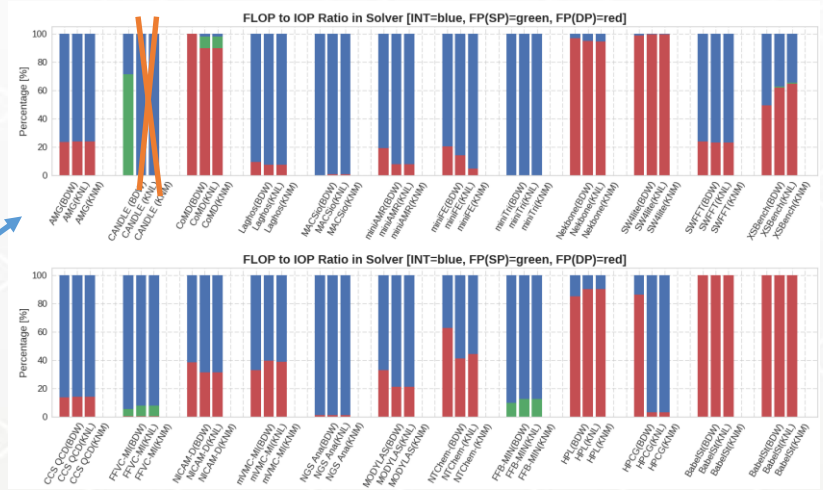
Things you hear…

“Wanna do HPC? Then you need fast FP64 matmul.”
--every HPC beginner class

More Flop/s ➔ more science?!

- Thanks to the (curse of) the TOP500 list, the **HPC community (and vendors) are chasing higher FP64 performance**, thru frequency, SIMD, more FP units, …
- **Motivation:**
  - **Less FP64 units**
    - **Saves power**
    - **Free chip area (ex: FP16)**
    - **Less divergence of "HPC-capable" CPUs from mainstream processors**
- Investigating many proxy-applications:
  - 

  - %FP32 vs. %FP64 vs. %Integer
    ➔ Integer (+DP) heavy (>50%; 16 of 22)
    ➔ Only 4 w/ FP32
    ➔ Only 1 mixed precision



J. Domke "Double-precision FPUs in High-Performance Computing: an Embarrassment of Riches?"

Compare Time-to-Solution in Solver

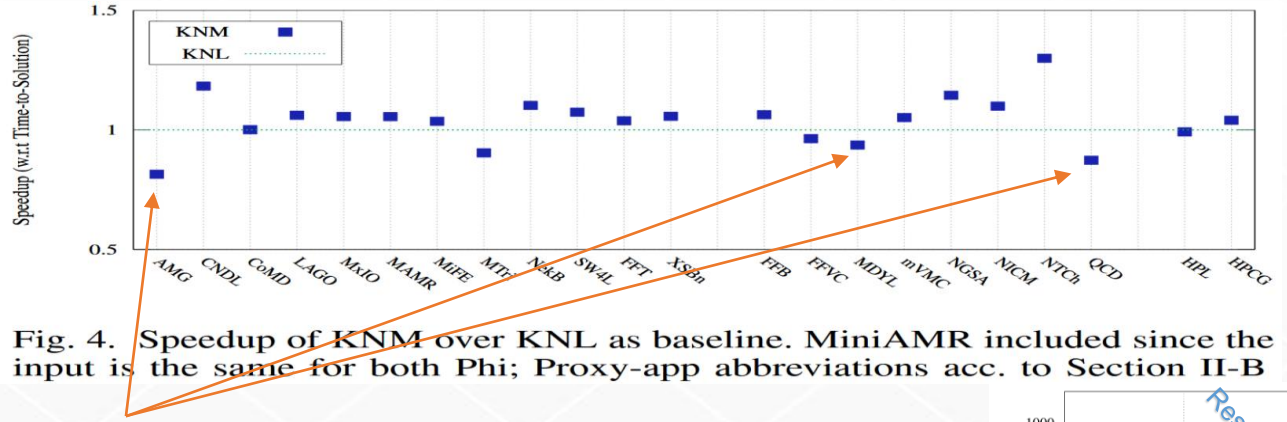
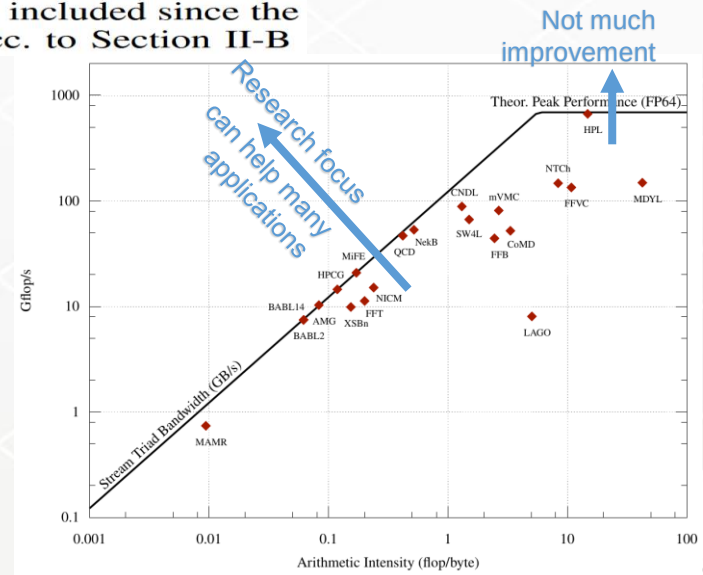


Fig. 4. Speedup of KNM over KNL as baseline. MiniAMR included since the input is the same for both Phi; Proxy-app abbreviations acc. to Section II-B

- **Only 3 apps seem to suffer from missing FP64 unit** (MiniTri: no FP; FFVC: only int+FP32)
- Options for **memory-bound** applications (almost all):
  - Invest in memory-/data-centric architectures
  - Move to FP32/mixed precision ➔ less memory pressure
- Options for **compute-bound** applications:
  - Brace for less FP64 units (driven by market forces) and less “free” performance (10nm, 7nm, 3nm, …then?)



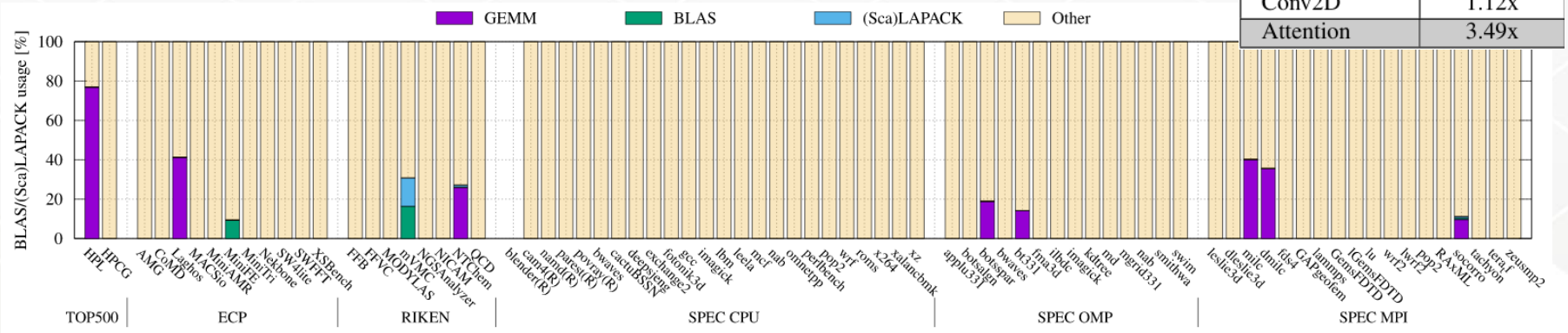
BLAS / GEMM utilization in HPC Applications

Computer simulations create the future

J. Domke "Matrix Engines for High Performance Computing: A Paragon of Performance or Grasping at Straws?"

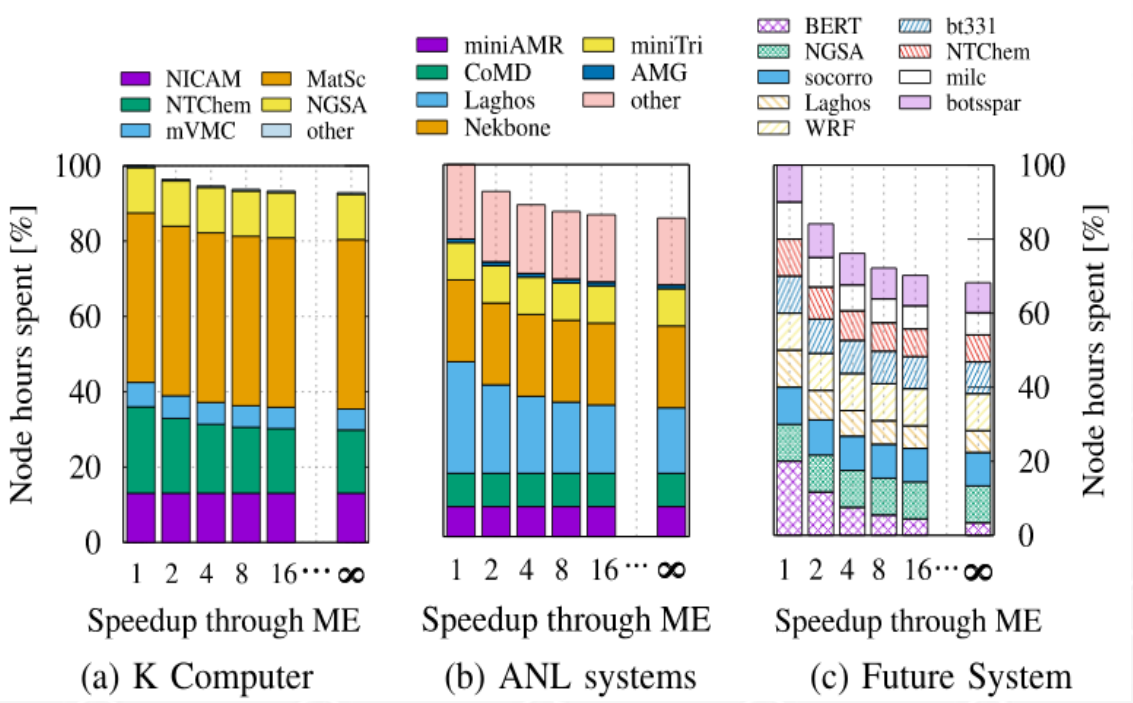
- Analyzed various data sources:
  - **Historical data from K computer**: only 53,4% of node-hours (in FY18) were consumed by applications which had GEMM functions in the symbol table
  - **Library dependencies**: only 9% of Spack packages have *direct* BLAS lib dependency (51.5% have indirect dependency)
  - **TensorCore benefit for DL**: up to 7.6x speedup for MLperf kernels
  - **GEMM utilization in HPC**: sampled across 77 HPC benchmarks (ECP proxy, RIKEN fiber, TOP500, SPEC CPU/OMP/MPI) and measured/profiled via Score-P and Vtune

Benchmark	Speedup
BERT	3.39x
Cosmoflow	1.16x
VGG16	1.71x
Resnet50	1.97x
DeepLabV3	1.75x
SSD300	1.78x
NCF	0.97x
GEMM	7.59x
GRU	3.67x
LSTM	5.69x
Conv2D	1.12x
Attention	3.49x



Estimated Benefit by MEs for HPC Centers

- Thought experiment: Assume we have/had GEMM units in past or future systems.
  - Known: node-hour by domain
  - Sample application with highest BLAS utilization
  - Estimate the node-hour reduction assuming different speedup by ME (2x–8x is realistic dep. on precision)
  - Future system includes 20% DL workloads, other science domains ~10% each



- Results w/ ideal conditions + 4x ME speedup: 5.3% less on K, 10.8% @ANL, 23.8% future system

➔ **HPC can utilize MEs when they come for free, but it's no magic bullet as for DL workloads**

➔ Explore more/other alternatives for Fugaku-next!

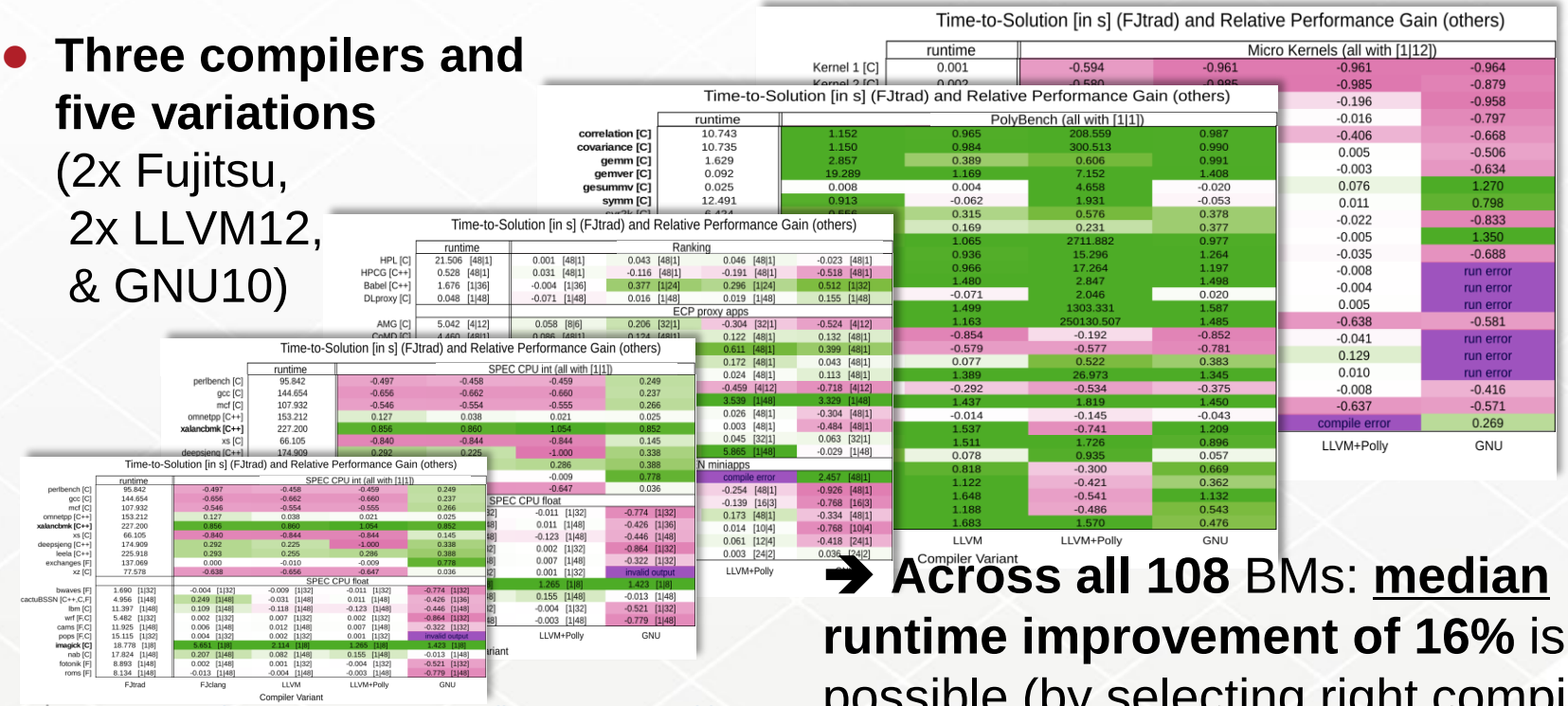
Things you hear…

“Porting an application to A64FX? Just use fcc and –Kfast.”

--Fujitsu

“Silver bullet” compiler choice for A64FX?

- **Performance portability** (x86→A64FX) **not easy** to achieve
- Testing **>100 Kernels and HPC Workloads** on Fugaku
- **Three compilers and five variations** (2x Fujitsu, 2x LLVM12, & GNU10)



➔ **Across all 108** BMs: **median runtime improvement of 16%** is possible (by selecting right compiler)

“Silver bullet” compiler choice for A64FX?

Conclusions:

- **C1**: recomm. usage model of **4 ranks and 12 threads often suboptimal**
- **C2**: **no “silver bullet” compiler** for A64FX (yet)
  - Dep. on situation, but some hint: **Fujitsu for Fortran codes,** and **GNU for integer-intensive apps,** and **any clang-based compilers for C/C++**
- **C3**: Twitter summary: “if Xeon is 70x faster than A64fx, suspect the compiler”

➔ Test all available compilers, and explore other rank/thread mappings!

Announcement:

- LLVM 13 incl. “classic” flang (source /home/apps/oss/llvm-v13.0.0/init.sh)
- SVE support still alpha ➔ expect even more performance with v14
- Potential roadblocks: Fujitsu’s MPI and SSL2

Things you hear…

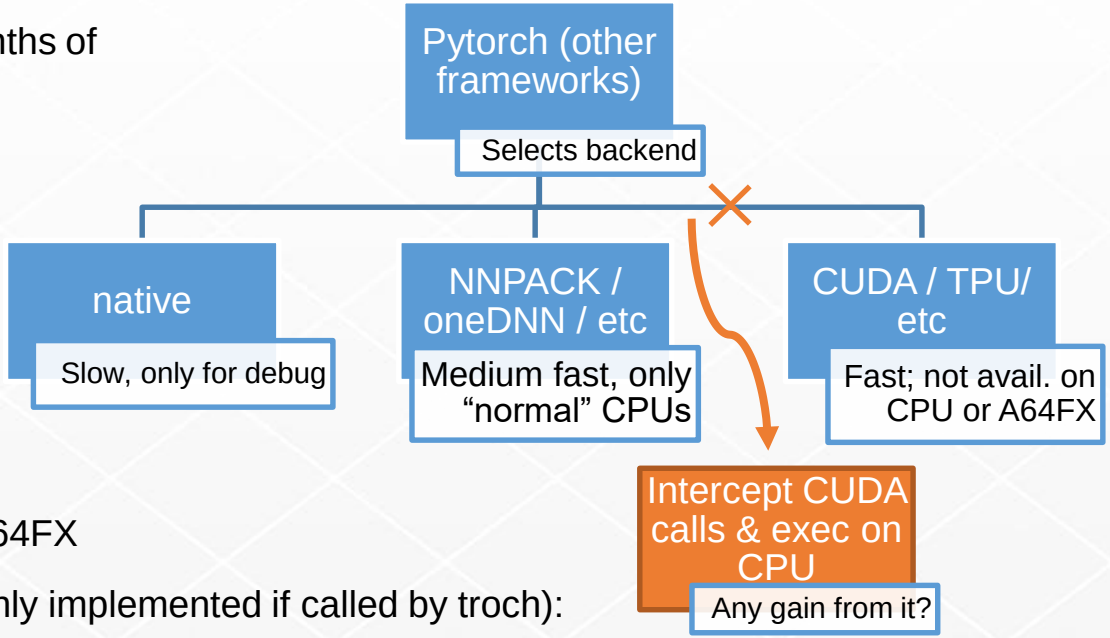
“Wanna do AI? Then you need NVIDIA’s GPUs.”
-- anonymous

“DL doesn’t run on CPU ‘cause of the sophisticated math.”
-- anonymous

“A64FX is more like a GPU than a CPU.”
--S. Matsuoka

DL4Fugaku – Replace CUDA RT & cuDNN?

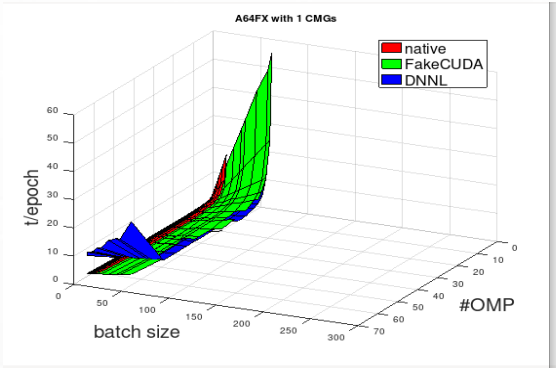
- Disadvantages of oneDNN approach:
  - Tedious to port to A64FX (months of engineering by Fujitsu)
  - Tuned for “normal” CPUs with assumption: Memory is slow
- **MocCUDA approach**
  - Fake availability of GPU attached to Fugaku nodes
  - Intercept CUDA calls
  - Execute CPU equivalent on A64FX

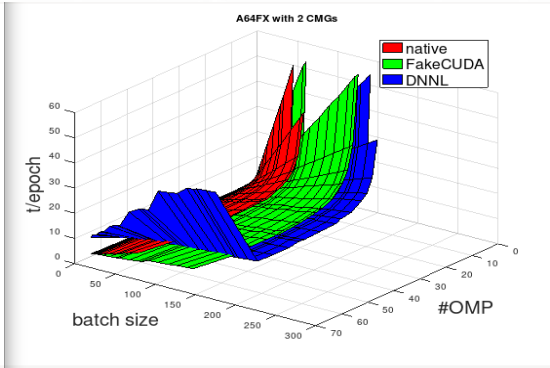


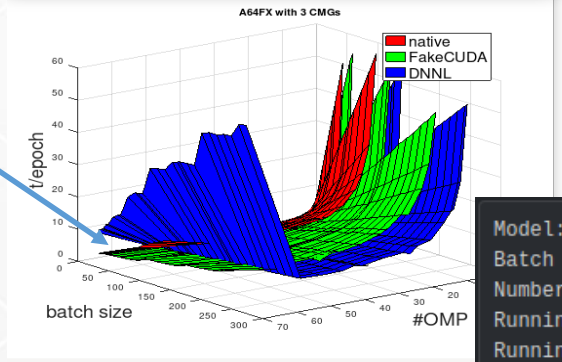
- **MocCUDA architecture** (func. only implemented if called by troch):
  - Wrapper library for CUDA runtime ➔ Easy
  - Wrapper libs for cuDNN (& cuBLAS) ➔ Medium hard (& trivial), no reference code available
  - Wrapper libs for native CUDA kernels (<<<…>>> in torch’s .cu files) ➔ Hard problem

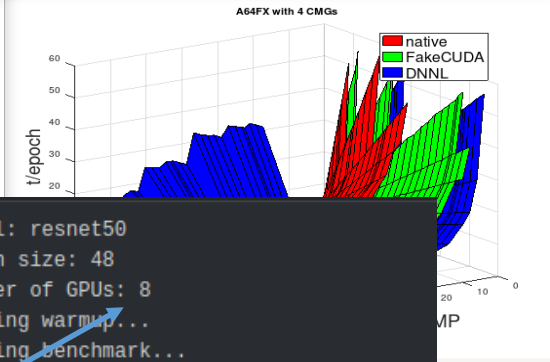
MocCUDA & Resnet50 Results

- Native implementation slow
- Native not scaling with batch size (OOM issues)
- oneDNNL has problem with #OMP > #cores
- **MocCUDA almost competitive (usually only 5%-20% slower)**
- **MocCUDA outperforms oneDNN (over 5x when #OMP > #core)**
- Support for other DL kernels and cuDNN functions can be added (only few missing to support MLPerf)
- **MocCUDA is still work-in-progress**
  - **Open issues: native CUDA kernels; SSL2 integration**
  - **Node-parallel training with Pytorch/Horovod on Fugaku**










```
Model: resnet50
Batch size: 48
Number of GPUs: 8
Running warmup...
Running benchmark...
Iter #0: 10.8 img/sec per GPU
Iter #1: 9.7 img/sec per GPU
Iter #2: 9.8 img/sec per GPU
Img/sec per GPU: 10.1 +-1.0
Total img/sec on 8 GPU(s): 80.8 +-8.1
```

Let’s clean up this mess …

Computer simulations create the future

“Octopodes to the rescue.”
--RIKEN & DOE

Fugaku Enhancement & Co-Design for Future

- Superseding current proxy-apps: ***Octopodes***
  - Downsides w/ Fiber/proxy-apps (s. Fugaku R&D)
  - On-going collaboration / brainstorming phase with DOE labs (position paper release in Apr.'22)
  - Set of highly-parameterizable, easily-amendable, MOTIF-like problem representations
  - **Common "language" between HPC users, system operators, co-designers, and vendors** to describe the to-be-solved scientific problems: What needs to be computed, and how it can be computed?

https://www.pinterest.fr/pin/145170787976811341/

➔ Apply ML to identify, parameterize, and categorize compute phases

S. Matsuoka, J. Domke, M. Wahib, A. Drozd, A. Chien, R. Bair, J. S. Vetter, J. Shalf
"Preparing for the Future –Rethinking Proxy Applications"
to appear in Computing in Science & Engineering

Usage of Octopodes for Co-Design

- "What needs & how can it be computed" not "Here is how you have to do it"
- For **performance modeling** of real workloads: identify compute phases which can be mapped to one or more Octopodes ➔ combine perf. model of the 'easier to understand' Octopodes ➔ approx. perf. model of full workloads
- For **vendors**:
  - Allowed tuning freedom for the Octopodes, i.e., changes of algo., implementation, integer/float. precision, data layout, etc., *as long as* intended result is the same
  - Accurately model consumer workloads ➔ Less over/under-selling of hardware
- **Porting** of user codes to new system:
  - Act as demonstrator for users to show how to port
  - ML/AI to identify phases can be used as helper for porting of real codes
- **Better suited for co-design tools**, e.g. compiler tests, regression testing, simulators (gem5/SST/CODES/…), quick "What-If" tools, etc.