Studying CPU and memory utilization of applications on Fujitsu A64FX and Nvidia Grace Superchip

Yan Kang
The Pennsylvania State University
United States
ybk5166@psu.edu

Sayan Ghosh
Pacific Northwest National Lab
United States
sayan.ghosh@pnnl.gov

Mahmut Kandemir
Penn State University
United States
mtk2@psu.edu

Andrés Márquez
Pacific Northwest National Lab
United States
andres.marquez@pnnl.gov

Abstract

ARM-based manycore CPU architectures are well-positioned to provide the rising memory throughput requirements of modern data intensive scientific applications in High Performance Computing (HPC). The Fujitsu A64FX CPU platform is based on the ARM v8.2A architecture, and is the processor of the flagship Japanese supercomputer - "Fugaku", which was previously ranked as the #1 supercomputer in the world according to the Top500 list. The Nvidia Grace superchip features 144 Neoverse V2 cores based on the ARMv9 architecture with 4x128b SVE2, providing exceptional computational power. The chip supports up to 480GB of memory, making it ideal for AI, machine learning, and scientific computing workloads. In this paper, we conduct a thorough performance exploration of a variety of parallel bandwidth-sensitive benchmarks and applications compiled with the native Fujitsu compiler on a Fugaku A64FX compute node and ARM (LLVM) Compiler on an NVIDIA Grace superchip compute node, engaging all the computational cores per cluster using OpenMP multithreading (assuming the cores can drive the available bandwidth). Our ultimate goals are to study the resource utilization of scientific applications and benchmarks on A64FX and Grace superchip, considering graph application scenarios (GAP Benchmark suite) and eleven application proxies from the Rodinia heterogeneous benchmark suite (considering domains such as Data Mining, Bioinformatics, Fluid Dynamics, Pattern Recognition, etc.). Through exhaustive performance monitoring, we quantify the resource utilization of diverse OpenMP-based HPC applications on both the Fujitsu A64FX and the Nvidia Grace Superchip platforms.

Keywords

Fujitsu A64FX, Fugaku, NVIDIA Grace superchip, Graph analytics, Rodinia benchmark suite, GAP benchmark

MEMSYS '24, September 30–October 03, 2024, Washington, DC, USA

ACM ISBN 979-8-4007-1091-9/24/09
https://doi.org/10.1145/3695794.3695813

ACM Reference Format:
Yan Kang, Sayan Ghosh, Mahmut Kandemir, and Andrés Márquez. 2024. Studying CPU and memory utilization of applications on Fujitsu A64FX and Nvidia Grace Superchip. In *The International Symposium on Memory Systems (MEMSYS '24), September 30–October 03, 2024, Washington, DC, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3695794.3695813

1 Introduction

Since the deployment of high-performance ARM multicore CPUs (i.e., Fujitsu A64FX) on formerly #1 supercomputer in the world, RIKEN Fugaku (circa 2020), there has been a number of high-end energy efficient processor design roadmaps with ARM-based CPUs (exhibiting 1 TB/s memory bandwidth, private caches and hundreds of CPU threads), among cloud/hyperscaler and data center product vendors. Most recently, NVIDIA™ released their first data center CPU, Grace, using a custom chip-to-chip interconnect (i.e., NVIDIA-C2C) to weave two CPU modules into a "superchip". We compare Fujitsu A64FX and NVIDIA Grace superchip in Fig. 1, listing the officially released performance numbers. Aside from being a larger CPU chipset (144 cores in Grace vs. 48 cores in A64FX), Grace superchip also offers a relatively large (117MiB/#NUMA node) shared Last Level Cache (LLC), which compared to A64FX's shared (per NUMA node or core group) L2 cache is 15× larger.

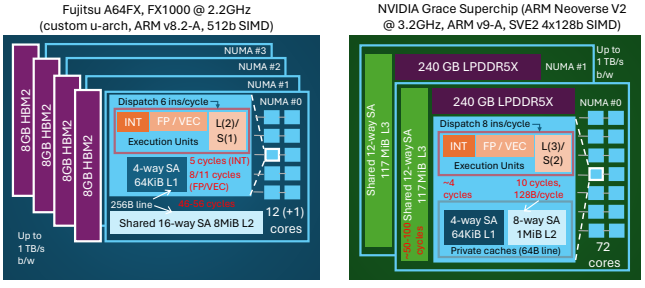

Figure 1: Fujitsu A64FX and NVIDIA Grace Superchip high-level components (using publicly available information).

A number of prior works have discussed the role of the ARM ecosystem in HPC, most recently laying out the architectural aspects of Fujitsu A64FX and NVIDIA Grace superchip, in the context of scientific applications [1, 9–12, 14, 17–19]. A key design

criteria of these "data center" CPUs are to enhance the throughput of memory-bound applications, which are ubiquitous in High Performance Computing (HPC). As such, both A64FX and Grace are capable of providing high instruction throughput, sustainable memory bandwidth and low latencies across the memory hierarchies (corroborated through popular benchmarks), yet performance footprints of myriad applications exhibit significant differences. In Fig. 2, we capture the memory latency spectrum for strided data transfers (as reported by LMBench [16]), corresponding to accesses across the memory hierarchy on "data center" CPUs (A64FX and Grace) vs. desktop CPU system, Apple M3. The impact of a relatively large LLC is evident in delaying the latency spikes as the data sizes increase (indicating data movement beyond private and shared caches).

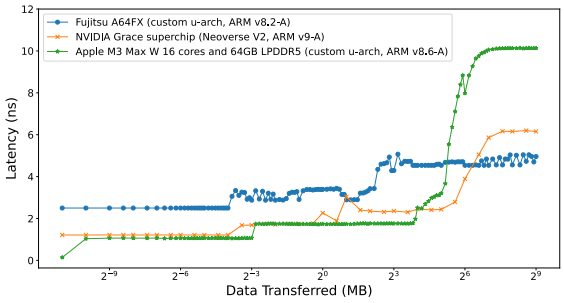

Figure 2: Access latencies (using LMBench, stride=128) across the memory hierarchy of Fujitsu A64FX, NVIDIA Grace and Apple M3.

In this paper, we consider both regular and irregular applications — while regular applications demonstrate fixed strides to access the memory blocks and may effectively engage the available cores to drive bandwidth, irregular scenarios might demonstrate relatively higher and unpredictable memory accesses and workload patterns. We have deliberately excluded FLOPs-intensive applications and benchmarks from our analysis, as scientific applications are leaning towards becoming more data-intensive and sparse, requiring broad set of optimizations to derive performance from future extreme-scale architectures. Recent research investigated time spent on Basic Linear Algebra Subprogram (BLAS) operations across several applications to not exceed 25% of the total execution time [20], therefore, optimizing third-party linear algebra libraries may not yield sustainable performance improvement. Our approach considers benchmark evaluation (using STREAM and graph neighborhood access benchmarks to study impact of contiguous vs. non-contiguous/ irregular writes) using various inputs followed by thorough profile-driven application analysis, using recommended compilers, options and profilers to extract the best performance from the individual platforms. Since the #cores vary between the platforms, we have a third configuration, which uses 48 cores on the NVIDIA Grace platform to keep parity with Fujitsu A64FX. Our studies indicate that most data intensive (especially irregular) applications benefit from a deeper memory hierarchy including a large LLC, and efficiencies in the load/store pipelines in processors can lead to free performance when applications are ported from A64FX platform to NVIDIA Grace superchip.

2 Benchmarks and Applications

We use standard HPC benchmarks and applications in our analysis, discussed in this section. The benchmarks incorporate the fundamental data-access patterns present in parallel applications. Most applications exhibit regular/contiguous access patterns, in which data movement happens across distinct memory locations at regular intervals, conducted by an iterative loop. On the other hand, graph applications involve graphs, which are mathematical structure analogous to a set comprising of points and lines; lines form "edges" to join two arbitrary points in the set, implying some type of relationship which can expressed via an attribute such as a scalar (or vector) weight over an edge. Usually, data corresponding to graphs is expressed by a hierarchical data structure, such as Compressed Sparse Row (CSR) format, which requires two levels of loop nesting to scan the edges corresponding to a vertex. This type of range indexing leads to irregularities in memory accesses, as the number of edges corresponding to a vertex may vary with diverse input graphs (edge distributions across a range of input graphs is shown in Fig. 3). We consider a multithreaded execution environ-

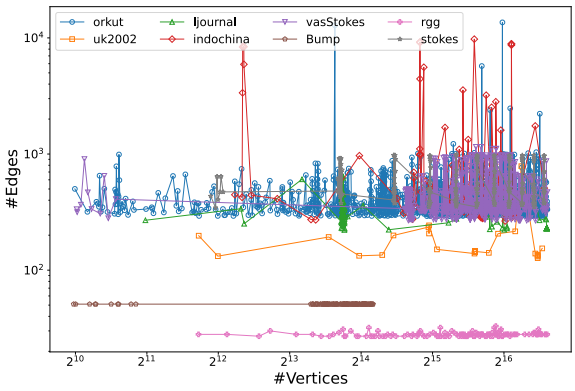

Figure 3: Top 100K #edges distribution across vertices of graphs.

ment (using OpenMP) in which a portion of the loop iterations are (statically) assigned to distinct processor threads. For our evaluations, we have not updated any code, benchmark or application, and present the results as-is. While it is practical to use a single metric or Figure-of-Merit (FOM) for a benchmark, for an application or even a mini-application, it is often intractable since different areas of an application may utilize different capabilities of the underlying processor. Therefore, we rely on visualizing performance profiles to understand the overall impact on the system across applications and inputs.

2.1 STREAM and Graph Neighborhood Accesses

STREAM benchmark [15] is commonly used to study the memory bandwidth of CPUs, by engaging all the processor cores to perform contiguous reads/writes to/from distinct memory locations, and performing simple arithmetic operations with the data. Additionally, we explore another benchmark to study irregular/noncontiguous

memory accesses, in the form of accessing the vertex neighborhoods of multifarious real-world graphs (comprising of a varied vertex-edge distributions), as depicted in Fig. 4, compared with access patterns in STREAM. As a result of the variation in the edges per vertex (owing to graph structure), if the loop over vertex is parallelized (which is often the case in real graph applications), there might be nontrivial load imbalance among the edges. Unlike

STREAM

```
#pragma omp parallel for
for i in LARGE_SIZED_ARRAY:
    a[i] = b[i]
```

Graph Neighborhood Access

```
#pragma omp parallel for
for i in #Vertices:
    for j in #Neighbors[i]:
        a[i] = b[j]
```

Figure 4: Distinct streaming access patterns: contiguous read/write in STREAM vs. non-contiguous read—contiguous write in Graph Neighborhood Access benchmark.

STREAM, memory access patterns of the graph neighborhood kernels primarily consists of traversing a doubly-nested loop mandated by the CSR representation of a graph. We use same set of graphs on different applications running on the experimental platforms, to study the effect of graph structure and application logic on the execution footprints. Influenced by STREAM, we develop parallel variants of *copy*, *add* and *max* kernels in the context of graphs (referred as graph neighborhood access kernels); the copy kernel copies the edge weights into an array, max collects the maximum of the edge weights among the vertex neighborhood into an array, whereas add accumulates the edge weights of a neighborhood copying it into an array. We generate a random geometric graph with good connectivity properties (dense clusters in the graph) in memory for benchmarking purposes. Like STREAM, we report the rate in MB/s for a kernel across a fixed number of iterations.

2.2 GAPBS and Rodinia

The GAP benchmark suite [3] consists of optimized parallel implementations of common graph algorithms supporting a wide variety of scientific applications. We use four complex graph kernels in our analysis: Breadth First Search (BFS), PageRank (PR), Connected Components (CC) and Betweenness Centrality (BC). Breadth First Search (BFS) performs level-by-level traversal of graphs. PageRank (PR) iteratively computes the popularity score of the vertices in a graph, by conducting repetitive sparse matrix-vector computations, such that the updated values are available immediately to be adjusted in the current iteration. The CC benchmark performs labeling of graph vertices according to their connected components, which is a set of vertices linked together by a path. Betweenness Centrality (BC) is about updating the scores of the vertices by computing shortest paths from a subset of vertices, usually implemented using several BFS traversals to approximate the scores.

Rodinia suite [5] comprises of several standalone mini-applications representing a variety of scientific application domains. We chose eleven mini-applications – Kmeans, Needleman-Wunsch (NW), HotSpot (HS), SRAD (versions V1 and V2), Back Propagation (BP), Leukocyte Tracking (LC), Stream Cluster (SC), CFD Solver (CFD), LU Decomposition (LUD) and Heartwall Tracking (HW); with appropriate input scenarios, as mentioned in Table 3. Kmeans is a traditional data mining application scenario that partitions datasets into a pre-defined set of clusters. Needleman-Wunsch (NW) is a dynamic programming method used to determine optimal genome sequence alignment. HotSpot (HS) consists of an iterative transient thermal simulation kernel that solves a series of differential equations for a block of temperatures. SRAD implements an image denoising technique based on partial differential equations representing a diffusion algorithm. Back Propagation is a classic Machine Learning algorithm for training weights of a neural network—comprising of the forward and backward phases. Leukocyte Tracking (LC) tracks White Blood Cells (WBC) in blood vessels; WBCs are first detected in an input video frame, which are subsequently tracked across rest of the frames, by computing specific gradients for each pixel. Stream Cluster (SC) performs online clustering for an input stream of points. CFD Solver (CFD) solves 3-D Euler equation for compressible flow, employing a finite volume formulation on unstructured grids. LU Decomposition (LUD) is a popular linear algebra algorithm for solving linear equations, by decomposing a matrix into lower and upper triangular matrices to maximize parallel efficiency. Heartwall Tracking (HW) uses image processing methods to track the shape of mouse heart over a sequence of fixed-resolution ultrasound images.

3 Evaluations and Analysis

Testbeds. Our testbed platforms are Fujitsu A64FX processor node of the Fugaku system [14] and NVIDIA Grace superchip node from Stony Brook University's Ookami cluster [4]. We use the recommended platform compilers and environment settings, as listed in Table 1. All the benchmarks and applications use OpenMP multithreading [7]. Since Grace superchip has 3× more CPU cores than Fujitsu A64FX, we additionally include a 48-threads configuration for Grace (selecting threads across the CPU sockets) in our baseline performance comparisons, to match with 48-thread runs of A64FX.

Table 1: Platform software details.

	A64FX	GRACE
Compiler	FUJITSU 4.10	ARMCLANG 24.04
Options	-Kfast -mcpu=a64fx -Kopenmp	-Ofast -mcpu=native -fopenmp
Affinity	OMP_PLACES=cores OMP_BIND=spread	OMP_PLACES=cores OMP_BIND=spread
Environment	XOS_MMM_L_ARENA_LOCK_TYPE=0 XOS_MMM_L_HPAGE_TYPE=hugetlbfs XOS_MMM_L_PAGING_POLICY= demand:demand:demand	N/A

Datasets and Input Parameters. We use various real-world graphs for the graph application scenarios in GAPBS. They are listed in Table 2. Typically, graphs are chosen such that the input size is greater than the Last Level Cache (LLC), to induce sufficient memory accesses. We use the same graphs on both A64FX and Grace platforms, although the latter can support larger graphs due to 16× greater main memory.

Table 3 lists the input arguments for the Rodinia benchmarks.

Table 2: Characteristics of the input graphs.

Graphs	Domain & Applied Benchmarks/Applications (GN : Graph Neighborhood Kernels)	#Vertices	#Edges	Max Deg.
com-orkut	social network(GN, GAPBS)	3M	234M	33K
rgg_n_2_24_s0	random geometric(GN, GAPBS)	16M	132M	40
uk-2002	web crawl(GN, GAPBS)	18M	584M	194K
LAW/ljournal-2008	social network(GN, GAPBS)	5M	79M	20K
indochina-2004	web crawl(GN)	7M	194M	256K
VLSI/vas_stokes_4M	semiconductor(GN)	4M	131M	1K
VLSI/stokes	semiconductor(GN)	11M	349M	2K
Janna/Bump_2911	reservoir simulation(GN, GAPBS)	3M	128M	195

Table 3: Rodinia kernels and input arguments.

Application	Domain	Problem Sizes
Kmeans	Data Mining	819200 data points, 34 features, 512 clusters
Needleman-Wunsch (NW)	Bioinformatics	81920 × 81920 data points with penalty of 10
HotSpot (HS)	Physics Simulation	1024 × 1024 data points with 1,500,000 iterations
Back Propagation (BP)	Pattern Recognition	65536 input nodes
SRAD_V1	Image Processing	502 × 458 image size with 100000 iterations
SRAD_V2	Image Processing	2048 × 2048 image size with 100000 iterations
Leukocyte Tracking (LC)	Medical Imaging	219 × 640 pixels/frame
Stream Cluster (SC)	Data Mining	655360 points, 256 dimensions
CFD Solver (CFD)	Fluid Dynamics	0.2M elements
LU Decomposition (LUD)	Linear Algebra	2048 × 2048 data points
Heart Wall Tracking (HW)	Medical Imaging	609 × 590 pixels/frame

Profiling metrics. We compare end-to-end execution times (in seconds) of the benchmarks and applications on a single Fujitsu A64FX and NVIDIA Grace superchip node. However, execution times alone does not convey the reasons behind the performance variations between application versions on the different platforms. Hence, we rely on systems profiling by studying the low-level performance events to understand the impact on the underlying system. Our profiling metrics consists of rates and unitless quantities characterizing specific CPU instruction overhead and load/stores across the memory hierarchy per NUMA node. Due to differences in the platform profilers, we are able to sample data at a finer granularity on A64FX platform (using Fujitsu Advanced Profiler, FAPP), as compared to userspace metrics collected from the `perf` Linux profiler on Grace superchip (enabling "–per-node" to aggregate per-NUMA-node measurements). Profiling metrics of A64FX and Grace are listed in Tables 4 and 5. We engage all the available cores during performance profiling.

On NVIDIA Grace superchip, we focus on two vital performance events: the volume of the *memory accesses* across the memory hierarchy (caches and main memory) and the CPU pipeline *stalls* (which corresponds to the wasted cycles, waiting for memory access operations or decoding instructions to work on data). Complex instructions can increase front-end stalls, whereas long-latency memory operations increase backend-stalls, throttling the front-end. FAPP on A64FX platform provides a more finer-grained details, such as integer and floating-point related cycles, and instructions break-up by functional units (i.e., integer, floating-point or prefetch).

3.1 Benchmarks

In this section, we compare the performance of STREAM (contiguous regular access data streams) with the graph neighborhood access kernels (demonstrates irregular noncontiguous data access pattern).

3.1.1 STREAM: . STREAM is memory-access intensive (we use 24GiB input array), greater than 50% of the time is spent on memory accesses (moving data through the cache hierarchy), as shown in Fig. 6, Grace profiles also corroborates this behavior of relatively high memory reads than writes, and low cross socket traffic (Fig. 7). Both FP and INT instruction pipelines are equally busy on A64FX across the CMGs (up to 7% of overall), suggesting most of the instructions are load/store.

Table 4: Fujitsu A64FX Profiling Metrics

% Busy	
FP	Busy rate for floating point operation pipelines
INT	Busy rate for integer operation pipelines
L1	Busy rate for primary cache
L2	Busy rate for secondary cache
MEM	Busy rate for memory
#Cache Misses	
LS	#Load/store instructions
L1M	L1 misses
L2M	L2 misses
Cycle Accounting (secs.)	
PF	Stalls due to prefetch port busy
INT	Stalls due to integer memory load
FP	Stalls due to floating-point memory load
INT(L2)	Stalls due to L2 cache access for integer load
FP(L2)	Stalls due to L2 cache access for floating-point load
Data Transfer (MB/s)	
own (R)	Throughput of reads within NUMA node
own (W)	Throughput of writes within NUMA node
otr (R)	Throughput of reads across NUMA nodes
otr (W)	Throughput of writes across NUMA nodes
#Instructions	
LS	#Load/store instructions
PF	#Prefetch instructions
FP	#Floating point math and conversion instructions
INT	#Integer operation instructions
Total	All above plus misc. (predicate, branch, etc.)

Table 5: NVIDIA Grace Profiling Metrics

CPU metrics	
INS	#CPU instructions
ST	#Stalls across CPU pipelines
STB	#Stalls in CPU backend (execution, memory) preventing instruction dispatch in frontend
STF	#Stalls in frontend due to lack of instructions to issue
SC	#Stalled cycles in backend (execution, memory) preventing instruction dispatch in frontend
STM	#Stalled cycles in backend due to memory load
Memory metrics	
LS	#Load/store counts
L1M	L1 misses
L2M	L2 misses
L3M	L3 misses
MA	#Memory accesses (due to load/stores)
MAR	#Memory reads
MAW	#Memory writes
MRA	#Memory accesses across NUMA nodes

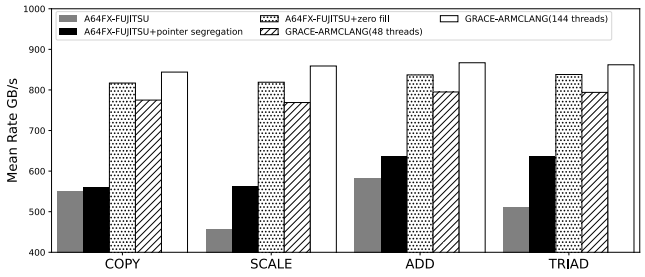

Figure 5: Performance of STREAM (GB/s, *higher* is better).

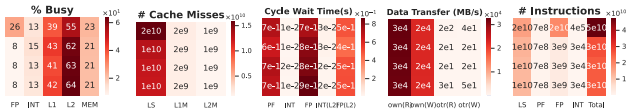

Figure 6: Performance events of STREAM on Fujitsu A64FX with Zero Filling (a row corresponds to measurements on a NUMA node, 4 NUMA nodes in A64FX).

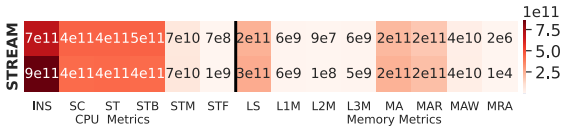

Figure 7: Performance events of STREAM on Grace (a row corresponds to measurements on a NUMA node, 2 NUMA nodes in Grace).

ADD/TRIAD moves about 30% more data than COPY/SCALE, so the bandwidth numbers are slightly better for ADD/TRIAD relatively, as shown in Fig. 5. We also observe that SCALE/TRIAD is about 20% worse than COPY/ADD on A64FX. Since pointers can potentially reference overlapping areas, some software pipelining optimizations are not turned on by default, causing the disparity. Upon passing `-Krestp=arg` (resolves pointer aliasing or segregation), such optimizations can be utilized (we only observed a major impact upon enabling this option for STREAM with the Fujitsu compiler, but not rest of the scenarios). Grace exhibits about 30% better overall bandwidth than A64FX: this can be due to the automatic enabling of the *write streaming mode* in Grace, which can prevent unnecessary cache reads when a cache line would be written anyways. In comparison, the Fujitsu compiler can enable the *write streaming mode* via the zero-fill ("-kzfill") compiler option, which zeroes a cache line using a special instruction (i.e., `DC ZVA`) upon detecting a streaming write operation. Enabling zero-fill, we observe maximum STREAM bandwidth on A64FX to be around 3% less than the full Grace system in Fig. 5. However, our prior work has shown limitations with implicitly or explicitly enabling zero-fill beyond simple streaming write scenarios [13]. Therefore, we exclude zero-fill from the rest of our evaluations.

3.1.2 Graph Neighborhood Access: Instead of considering contiguous reads and writes for bandwidth measurement (best case, i.e., STREAM), the graph neighborhood access benchmark considers traversing an entire graph in parallel (each thread owns fixed number of iterations over the vertices of a graph and performs some operations in proportion to the #edges, edge-distribution across vertices being dissimilar, see §2.1). This type of traversal leads to irregular/noncontiguous reads. Inspired from STREAM, we consider COPY, ADD and MAX variants of the graph benchmark which works on the edges of a graph incurring the same number of arithmetic operations per edge-iteration as STREAM. The baseline performance in Fig. 8 indicates major performance variations across the graphs, attributed solely due to the structure of the graphs (as shown in Fig. 3). Also, comparing the STREAM profiles with graph neighborhood access (for e.g., Figures 6 vs. 9), the impact of load imbalance is evident: integer/floating-point stalls and instructions are notably higher than those in STREAM indicating load imbalance and complex indexing. Differences between A64FX and Grace are also apparent: in a few cases we observe the performance of the graph neighborhood benchmark on Grace to be comparable with STREAM, whereas on A64FX, the best outcome was about 50% of the STREAM bandwidth. However, we observe similar performances on Grace and A64FX for two graphs—ljournal and indochina, both depicting highly irregular edge distributions (Fig. 3) despite relatively high maximum degrees. Irregular degree distributions can lead to severe load imbalances.

The A64FX performance profiles in Fig. 9 can explain the observed performance differences between ljournal/indochina and the rest. Both ljournal/indochina exhibits higher stalls due to prefetching, increased load/store operations and relatively high cross-NUMA-node traffic. These observations also match the Grace performance events, as shown in Fig. 10, which depicts significantly high memory accesses compared to other inputs.

For uk-2002 input, both A64FX and Grace demonstrate better performance relative to other inputs; uk-2002 demonstrates relatively uniform degree distribution (standard deviation of #edges/vertex is about 13), and as a result a balanced workload across the NUMA nodes (as shown in Figures 9 and 10), leading to better stalls profile (stalls indicate waiting for data either from memory or cache, lower is better); inputs bump and rgg also demonstrates better bandwidth on Grace, due to uniform degree distributions. Additionally, bump demonstrates significantly better performance on Grace as compared to A64FX. On Grace, bump demonstrates the least cross-NUMA domain memory accesses among the other inputs, which is not observed in A64FX. Another reason behind greater than about 30% improved performance on Grace compared to A64FX as shown in Fig. 8 can be attributed to the large capacity of LLC (234MiB). In Fig. 6, we observe relatively comparable L1 and L2 cache misses, whereas on Grace, L2 cache misses can be seen to be two or three orders of magnitude lower in comparison (Fig. 10), indicating significant data reuse (L3 is unified, so the reported numbers might be higher than private/separate L2).

3.2 Application Scenarios

We discuss the application evaluations in this section, starting with the GAP benchmark suite in §3.2.1, which consists of optimized reference implementations of several key graph algorithms. Next,

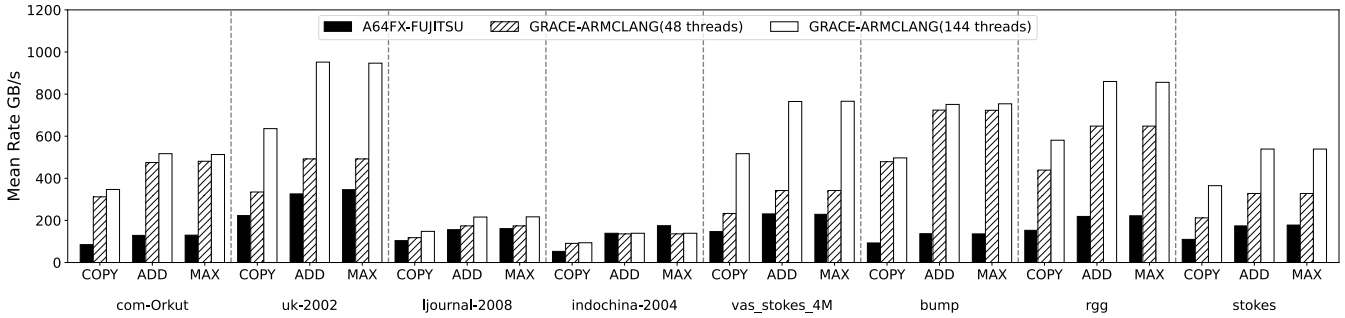

Figure 8: Performance of graph neighborhood kernels (GB/s, *higher* is better) across compilers and graphs.

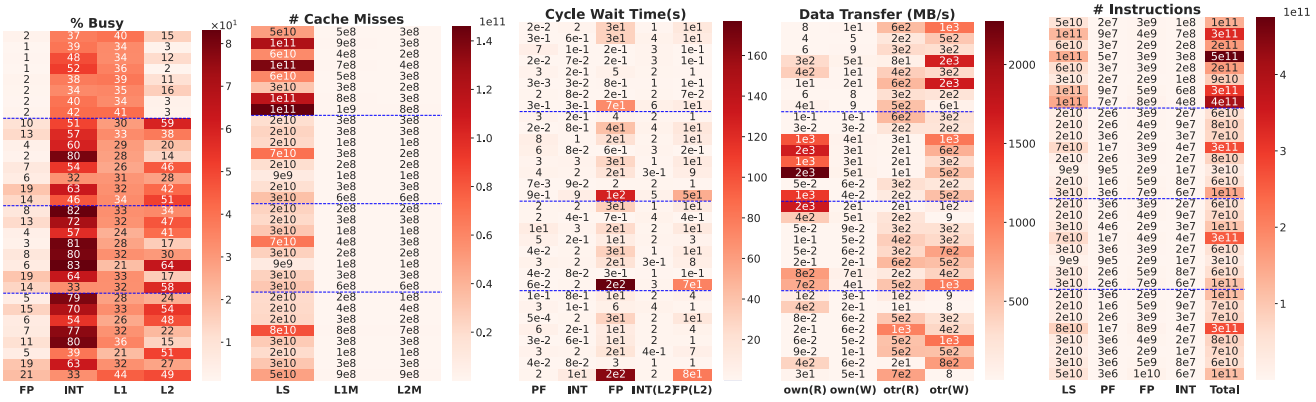

Figure 9: Performance events of graph neighborhood benchmarks on Fujitsu A64FX for various graphs (a row corresponds to measurements on a NUMA node, 4 NUMA nodes in A64FX).

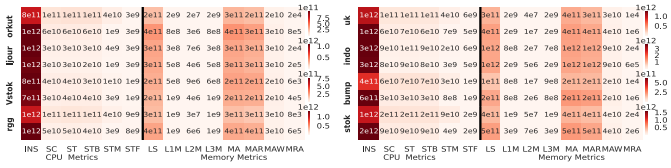

Figure 10: Performance events of graph neighborhood benchmarks on NVIDIA Grace superchip for various graphs (a row corresponds to measurements on a NUMA node, 2 NUMA nodes in Grace superchip).

in §3.2.3 we discuss several benchmarks from the Rodinia suite, covering diverse application domains.

3.2.1 GAP Benchmark Suite. The GAP benchmark suite [3] aims to standardize benchmarking of graph analytics, by providing a specification and high-performance modern C++-based reference shared-memory implementations of common graph algorithms with a wide variety of applications. We use four graph kernels from the benchmark: Breadth First Search (BFS), PageRank (PR), Connected Components (CC) and Betweenness Centrality (BC). The baseline results are shown in Fig. 11 against five input graphs (Table 2); increasing the #threads does not improve the performance significantly, but we still observe about up to 10× better performance on Grace than A64FX.

GAP implements an optimal "direction-optimizing" method of BFS [2] which combines the classic top-down BFS with a bottom-up step that allows a vertex to check whether its parent is in the "frontier" list of unvisited vertices (instead of a vertex checking its adjacent vertices, akin to a parent looking for a child). The amount of parallelism is proportional to the size of the frontier, as a result BFS on certain graphs can suffer from starvation with increased number of threads. PageRank (PR) iteratively computes the popularity score of the vertices in a graph by implementing sparse matrix-vector computations, employing OpenMP nested parallelism which may not be supported by the underlying runtime due to overheads associated with oversubscription, leading to minor improvements between 48 and 144 threads on Grace for various inputs. The CC benchmark performs parallel labeling of the graph vertices according to their connected components, which is a set of vertices linked together by a path. Betweenness Centrality (BC) is about updating the scores of the vertices by computing shortest paths from a subset of vertices. This is usually implemented using multiple BFS traversals to approximate the scores, and uses atomic operations to track possible paths.

Profiling analysis of GAPBS considers different NUMA nodes and input graphs, measurements of different graphs are stacked per NUMA node (depicted by broken blue lines), as shown in Figures 12, 13, 14 and 15. The A64FX profiles indicates most of the

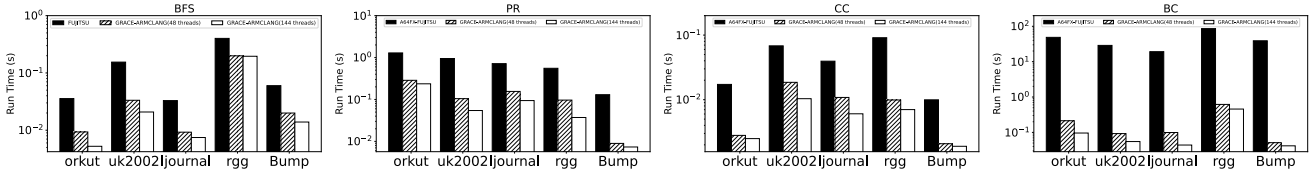

Figure 11: Performance of GAP benchmarks (GAPBS) on A64FX and Grace platforms.

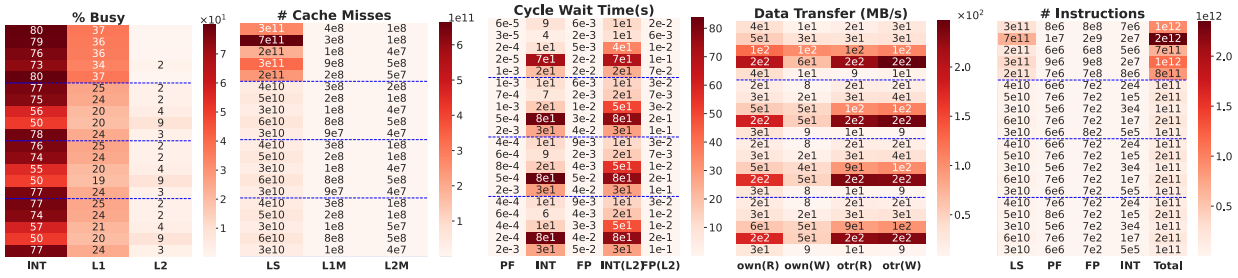

Figure 12: Performance events of GAPBS(BFS) on Fujitsu A64FX for several graphs (broken blue lines indicates NUMA node).

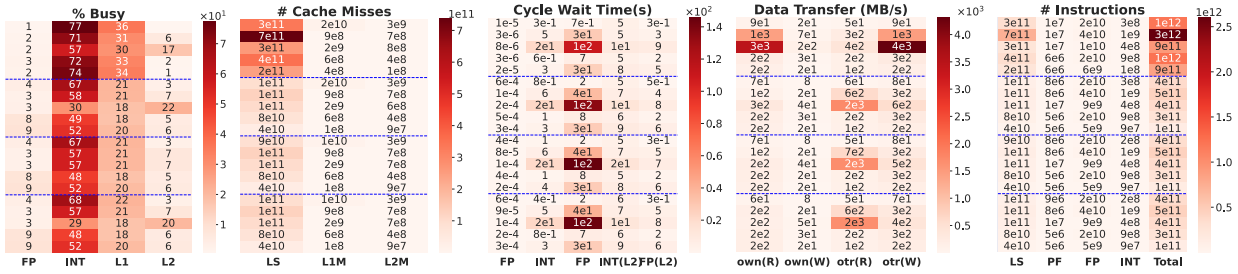

Figure 13: Performance events of GAPBS(PR) on Fujitsu A64FX for several graphs (broken blue lines indicates NUMA node).

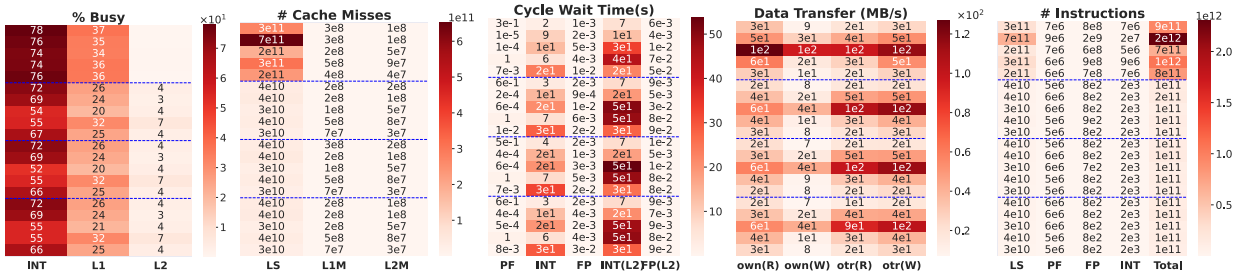

Figure 14: Performance events of GAPBS(CC) on Fujitsu A64FX for several graphs (broken blue lines indicates NUMA node).

overhead in integer operations (despite floating-point instruction throughput being higher) and traffic across the NUMA nodes. Accordingly, corresponding Grace profiles (Figures 16 and 17) also points to the traffic across the NUMA nodes and relatively high backend stalled cycles. However, the data movement across NUMA nodes for A64FX is higher than that of Grace (in the A64FX profiles, data transfer is represented in MB/s, so the numbers should be multiplied with 10^6 to roughly compare with corresponding Grace memory metrics), and relatively less misses in the cache hierarchy led to improved end-to-end performance. Also, the cache behavior and the prefetch instructions of GAPBS are comparable to the graph neighborhood benchmarks, due to same input graphs. Comparing the Grace profiles of GAPBS with graph neighborhood benchmarks (Figures 16 and 17, vs. 10), we observe comparable volumes of memory accesses (reads), nearly all stalls are backend stalls (waiting on memory loads). Therefore, having better latencies across the memory hierarchy (about 50% better in Grace relative to A64FX, Fig. 2) is beneficial for graph workloads.

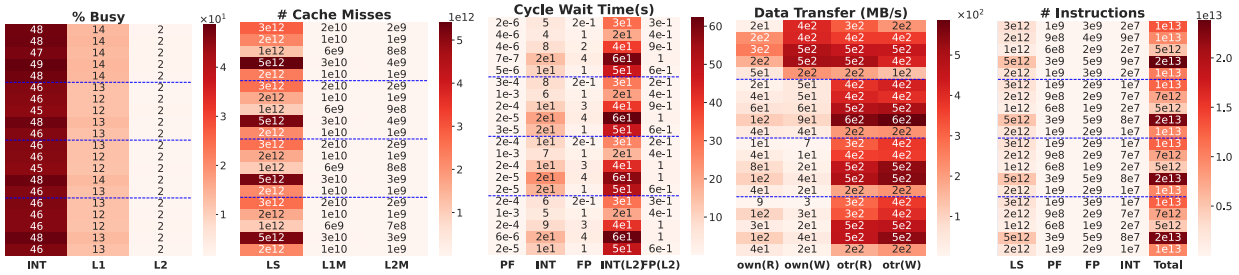

Figure 15: Performance events of GAPBS(BC) on Fujitsu A64FX for several graphs (broken blue lines indicates NUMA node).

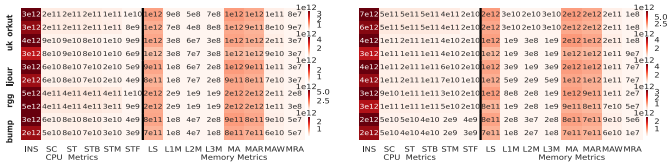

Figure 16: Performance events of GAPBS BFS and PR on NVIDIA Grace for several graphs.

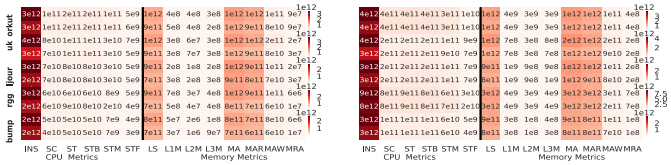

Figure 17: Performance events of GAPBS CC and BC on NVIDIA Grace for several graphs.

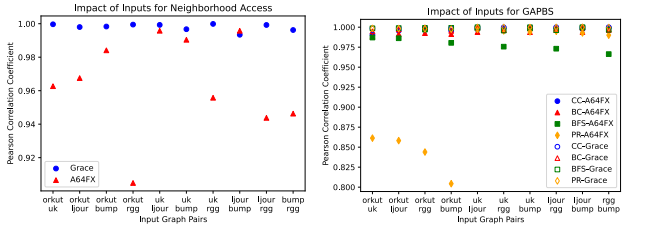

Figure 18: Impact of graph inputs on the Grace and A64FX platforms across Neighborhood Access benchmarks (left) and GAPBS (right). We compare the performance events datasets (collected from `Perf`) for distinct input graph pairs via the Pearson correlation coefficient which measures linear relationships between the sets (0 implies no correlation, whereas 1 indicates positive correlation).

3.2.2 Impact of input graphs. The inherent structure of the graphs can impact the overall performance, due to the load imbalance across the available threads. In Fig. 18, we calculate the Pearson correlation coefficient [6] between the subsequent CPU/memory access profiles (considering the same events collected using `Perf` on both the platforms as listed on Table 5, but does not distinguish NUMA regions) of Neighborhood Access and GAPBS scenarios across pairwise input graphs. While quantifying the impact of inputs via simple linear relationships can be problematic, but the goal here is to compare the performance footprints of multifarious input graphs across the platforms (engaging all available processing cores/threads). The results indicate a significant diversity between inputs (for both benchmarks and applications) on the A64FX platform particularly, as compared to Grace. Aside from the performance differences due to the hardware capabilities, relatively a smaller number of threads on A64FX (48 vs. 144 on Grace) accentuates the load imbalance (each thread owns a fixed number of iterations resembling vertices, but dissimilar number of edges). As the number of vertices per thread reduces (due to greater processing cores on Grace), so does the load imbalance. This effect can be observed in Fig. 11 by comparing the respective patterns of 48 vs. 144 threads performance of GAPBS applications under different inputs.

3.2.3 Rodinia benchmarks. Rodinia heterogeneous suite [5] comprises of several standalone benchmarks spanning a range of application motifs, as discussed in §2.2. The baseline result is presented in Fig. 20; aside from StreamCluster (SC), we observe about up to 10× improvement in end-to-end performance on the Grace platform, as compared to A64FX. Like the GAP kernels, we observe performance saturation on Grace using all of the available 144 cores (see Fig. 20). Also unlike the graph benchmarks and mini-applications, we observe different patterns in the A64FX performance profiles (Fig. 19), such as very high reuse (due to loop blocking), higher floating-point operation overheads (compared to integer operations) and stalls due to memory accesses and instruction execution. However, both GAPBS and Rodinia exhibits relatively similar data transfer volumes (especially writes) across the NUMA regions.

On Grace, several Rodinia kernels such as CFD, SRAD and SC exhibits high stalls, limiting the parallel efficiency, as shown in Fig. 21. Due to the high capacity of LLC, we also observe maximum reuse (and unbalanced memory access overhead) and minimal stalls in LUD, depicting multiple orders of magnitude speedup on Grace relative to A64FX. We also observe relatively high remote memory accesses for some of the benchmarks such as HS, SC and CFD. A64FX partitions the cache into sectors such that misses are only restricted to a particular "sector", with the intention to improve the overall cache misses due to premature eviction of an entire line. We observe a minor impact of the sector cache in reducing the LLC cache misses by about 5% for some of the compute intensive applications in Rodinia (e.g., BP, CFD, etc.); for the graph workloads, we do not observe any particular evidence of improvement.

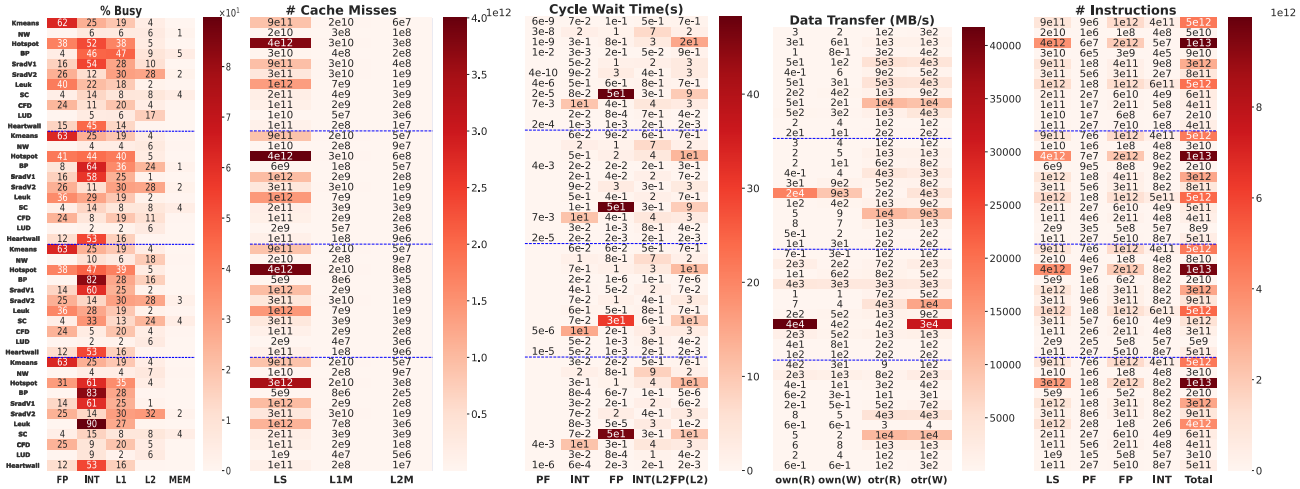

Figure 19: Performance events of Rodinia mini-applications (per line) on Fujitsu A64FX (broken blue lines indicates NUMA node, for e.g., a row on every NUMA node corresponds to a specific application).

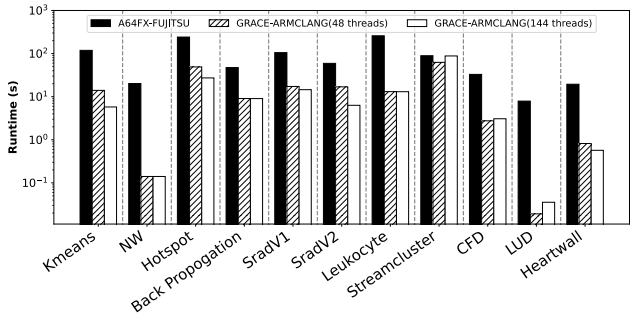

Figure 20: Performance of Rodinia mini-applications on A64FX and Grace platforms.

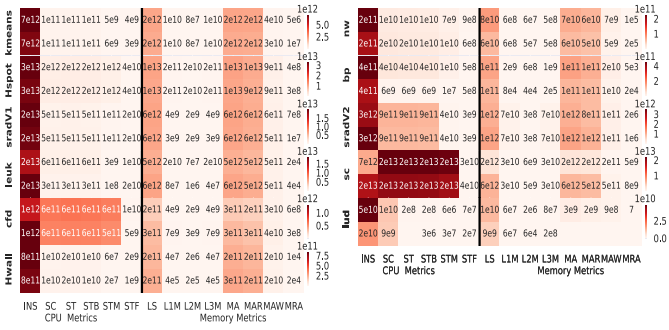

Figure 21: Performance events of Rodinia on NVIDIA Grace.

4 Concluding Remarks

In this paper, we provide a thorough quantitative analysis of various HPC applications on two contemporary ARM-based multiprocessor platform — Fujitsu A64FX and NVIDIA Grace superchip. We observed competitive performance on regular benchmarks such as STREAM across the platforms, but for more irregular benchmarking scenarios, NVIDIA Grace outperformed Fujitsu A64FX by 30–50% for a variety of input graphs. Moreover, considering more involved data intensive application scenarios, the performance gap between A64FX and Grace were significant, up to orders of magnitude (with no code changes). Our high-level takeaways are as follows:

- Regular application patterns such as streaming writes or nontemporal stores might benefit from platform optimizations such as write-allocate evasion or elimination (available on both A64FX and Grace), however they are not likely on the critical path in applications (primarily used for initializing data structures), so the outcome will depend on the specific application situation. This option can detect simple benchmarking patterns, such as STREAM, so more complex and irregular workload analysis is mandatory.
- For irregular workloads such as graph analytics, the structure of the input graph is quite relevant in determining the throughput. However, more available threads can also reduce the overall load imbalance due to the graph structure.
- Even irregular applications have regular patterns, having high-capacity LLC is beneficial in optimizing reuse.
- A number of applications depend on integer pipeline throughput, optimizing integer pipeline is as important as floating-point.
- For a number of data intensive applications, data exchange traffic across the NUMA nodes can be significant, therefore, sustainable bandwidth between NUMA nodes is crucial.
- A number of applications are unable to drive the available bandwidth, using all the available cores might lead to starvation and impact the parallel efficiency. This is partly due to the algorithm implementation and limitations in the underlying programming model. Adopting modern performance portability abstractions [8] in developing memory bound applications can potentially enhance the parallel efficiency.

Overall, our results quantitatively reinforces the immense potential of performance improvement for scientific applications across contemporary high-performance ARM data center CPU platforms.

Acknowledgments

This work was supported by the US DOE Office of Science project "Advanced Memory to Support Artificial Intelligence for Science" at PNNL. PNNL is operated by Battelle Memorial Institute under Contract DE-AC05-76RL01830. Research at PSU is supported by National Science Foundation (NSF) grants #2211018, #1931531, and #2008398.The authors would like to thank Stony Brook Research Computing and Cyberinfrastructure, and the Institute for Advanced Computational Science at Stony Brook University for access to the Ookami computing system.

References

[1] Fabio Banchelli, Joan Vinyals-Ylla-Catala, Josep Pocurull, Marc Clascà, Kilian Peiro, Filippo Spiga, Marta Garcia-Gasulla, and Filippo Mantovani. 2024. NVIDIA Grace Superchip Early Evaluation for HPC Applications. In *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region Workshops*. 45–54.

[2] Scott Beamer, Krste Asanovic, and David Patterson. 2012. Direction-optimizing breadth-first search. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–10.

[3] Scott Beamer, Krste Asanović, and David Patterson. 2015. The GAP benchmark suite. *arXiv preprint arXiv:1508.03619* (2015).

[4] Andrew Burford, Alan Calder, David Carlson, Barbara Chapman, Firat Coskun, Tony Curtis, Catherine Feldman, Robert Harrison, Yan Kang, Benjamin Michalowicz, et al. 2021. Ookami: Deployment and Initial Experiences. (2021), 1–8.

[5] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE international symposium on workload characterization (IISWC)*. Ieee, 44–54.

[6] Israel Cohen, Yiteng Huang, Jingdong Chen, Jacob Benesty, Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. 2009. Pearson correlation coefficient. *Noise reduction in speech processing* (2009), 1–4.

[7] Leonardo Dagum and Ramesh Menon. 1998. OpenMP: an industry standard API for shared-memory programming. *IEEE computational science and engineering* 5, 1 (1998), 46–55.

[8] Tom Deakin, Simon McIntosh-Smith, James Price, Andrei Poenaru, Patrick Atkinson, Codrin Popa, and Justin Salmon. 2019. Performance portability across diverse computer architectures. In *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*. IEEE, 1–13.

[9] Jens Domke. 2021. A64FX–Your Compiler You Must Decide!. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 736–740.

[10] Anne C Elster and Tor A Haugdahl. 2022. Nvidia hopper gpu and grace cpu highlights. *Computing in Science & Engineering* 24, 2 (2022), 95–100.

[11] Jonathon Evans. 2022. Nvidia Grace. In *2022 IEEE Hot Chips 34 Symposium (HCS)*. IEEE Computer Society, 1–20.

[12] Adrian Jackson, Andrew Turner, Michèle Weiland, Nick Johnson, Olly Perks, and Mark Parsons. 2019. Evaluating the arm ecosystem for high performance computing. In *Proceedings of the Platform for Advanced Scientific Computing Conference*. 1–11.

[13] Yan Kang, Sayan Ghosh, Mahmut Kandemir, and Andrés Marquez. 2024. Impact of Write-Allocate Elimination on Fujitsu A64FX. In *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region Workshops*. 24–35.

[14] Satoshi Matsuoka. 2021. Fugaku and A64FX: the first exascale supercomputer and its innovative ARM CPU. In *2021 Symposium on VLSI Circuits*. IEEE, 1–3.

[15] John D McCalpin. 1995. Stream benchmark. *Link: www. cs. virginia. edu/stream/ref. html# what* 22 (1995), 7.

[16] Larry W McVoy, Carl Staelin, et al. 1996. Lmbench: Portable tools for performance analysis.. In *USENIX annual technical conference*. San Diego, CA, USA, 279–294.

[17] Benjamin Michalowicz, Eric Raut, Yan Kang, Tony Curtis, Barbara Chapman, and Dossay Oryspayev. 2021. Comparing OpenMP Implementations with Applications Across A64FX Platforms. In *International Workshop on OpenMP*. Springer, 127–141.

[18] Tetsuya Odajima, Yuetsu Kodama, Miwako Tsuji, Motohiko Matsuda, Yutaka Maruyama, and Mitsuhisa Sato. 2020. Preliminary performance evaluation of the Fujitsu A64FX using HPC applications. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 523–530.

[19] Mitsuhisa Sato, Yutaka Ishikawa, Hirofumi Tomita, Yuetsu Kodama, Tetsuya Odajima, Miwako Tsuji, Hisashi Yashiro, Masaki Aoki, Naoyuki Shida, Ikuo Miyoshi, et al. 2020. Co-design for a64fx manycore processor and" fugaku". In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.

[20] Harry Waugh and Simon McIntosh-Smith. 2020. On the use of BLAS libraries in modern scientific codes at scale. In *Driving Scientific and Engineering Discoveries Through the Convergence of HPC, Big Data and AI: 17th Smoky Mountains Computational Sciences and Engineering Conference, SMC 2020, Oak Ridge, TN, USA, August 26-28, 2020, Revised Selected Papers 17*. Springer, 67–79.