Bit Contiguous Memory Allocation for Processing In Memory

John D. Leidel
Tactical Computing Laboratories
McKinney, Texas 75070
jleidel@tactcomplabs.com

ABSTRACT

Given the recent resurgence of research into processing in or near memory systems, we find an ever increasing need to augment traditional system software tools in order to make efficient use of the PIM hardware abstractions. One such architecture, the Micron In-Memory Intelligence (IMI) DRAM, provides a unique processing capability within the sense amp stride of a traditional 2D DRAM architecture. This accumulator processing circuit has the ability to compute both horizontally and vertically on pitch within the array. This unique processing capability requires a memory allocator that provides physical bit locality in order to ensure numerical consistency. In this work we introduce a new memory allocation methodology that provides bit contiguous allocation mechanisms for horizontal and vertical memory allocations for the Micron IMI DRAM device architecture. Our methodology drastically reduces the complexity by which to find new, unallocated memory blocks by combining a sparse matrix representation of the array with dense continuity vectors that represent the relative probability of finding candidate free blocks. We demonstrate our methodology using a set of pathological and standard benchmark applications in both horizontal and vertical memory modes.

CCS CONCEPTS

• **Software and its engineering → Software libraries and repositories**; • **Computer systems organization** → *Single instruction, multiple data*; *Heterogeneous (hybrid) systems*;

KEYWORDS

memory allocation; processing near memory; vector processor; SIMD

ACM Reference format:
John D. Leidel. 2017. Bit Contiguous Memory Allocation for Processing In Memory. In *Proceedings of MCHPC'17: Workshop on Memory Centric Programming for HPC, Denver, CO, USA, November 12–17, 2017 (MCHPC'17),* 9 pages.
DOI: 10.1145/3145617.3145618

1 INTRODUCTION

Given the recent technological shift in high performance, stacked and embedded memory technologies, we have experienced a shift back to research efforts associated with processing near and processing in memory. These efforts have been generally focused on two hardware methods: stacking logic layers under DRAM device layers or modifying the fundamental DRAM logic circuit.

The Micron In-Memory Intelligence (IMI) device technology [8] is a prime example of the latter alternative DRAM logic circuit. The IMI device is built upon a series of simple, bit-serial computing elements placed on pitch below each sense-amplifier. Each bit-serial computing element is capable of basic logical functions which can be combined with the data movement capabilities of the sense amp to form basic arithmetic operations within the array.

However, unlike traditional DRAM devices, these bit-serial operations rely upon a distinct physical locality between the individual bits of an element. In this manner, bits within an element must be physically co-located within the IMI array in order to maintain numerical consistency in boolean and arithmetic operations. This requirement of physical bit continuity is orthogonal to the standard memory addressing methods found in traditional DRAM memory subsystems. Further, traditional memory allocation schemas, such as those found in GNU's *malloc* API, enforce memory allocation that is interleaved across banks in order to enhance performance. This work describes a bit-contiguous memory allocation methodology built specifically for the Micron IMI device architecture that explicitly allocates memory blocks per the physical locality rules defined therein.

The main contributions of this work are as follows. We introduce a novel main memory allocation methodology for a state of the art processing in memory (PIM) architecture. Given the novelty of the processing architecture's ability to compute values on pitch in the sense amp array, our methodology enforces strict bit continuity in its allocation schema in order to preserve numerical consistency when computing values across bit lines. Finally, we demonstrate that our approach provides a substantially lower time complexity when searching for unallocated, bit-contiguous blocks against the standard *MxN* approach.

The remainder of this work is organized as follows. Section 2 presents previous works in processor near memory and memory allocation methodologies. Section 3 provides an overview of the Micron IMI architecture and a rudimentary explanation of its programming model. Section 4 describes our approach and associated implementation. Section 5 provides an evaluation of our approach and Section 6 provides our final conclusions.

2 PREVIOUS WORK

Recent analysis of large-scale data center workloads has revealed that a large percentage of the total workload is actually spent allocating and managing memory. As result, several research efforts have sought to accelerate memory allocation for large scale data centers using modifications to the core SoC [11]. Further, additional

MCHPC'17, Denver, CO, USA

DOI: 10.1145/3145617.3145618

Beyond 16GB: Out-of-Core Stencil Computations

István Z Reguly
Faculty of Information Technology and Bionics,Pázmány Péter Catholic University
Budapest, Hungary
reguly.istvan@itk.ppke.hu

Gihan R. Mudalige
Department of Computer Science, University of Warwick
Coventry, UK
g.mudalige@warwick.ac.uk

Michael B. Giles
Maths Institute, University of Oxford
Oxford, UK
mike.giles@maths.ox.ac.uk

ABSTRACT

Stencil computations are a key class of applications, widely used in the scientific computing community, and a class that has particularly benefited from performance improvements on architectures with high memory bandwidth. Unfortunately, such architectures come with a limited amount of fast memory, which is limiting the size of the problems that can be efficiently solved. In this paper, we address this challenge by applying the well-known cache-blocking tiling technique to large scale stencil codes implemented using the OPS domain specific language, such as CloverLeaf 2D, CloverLeaf 3D, and OpenSBLI. We introduce a number of techniques and optimisations to help manage data resident in fast memory, and minimise data movement. Evaluating our work on Intel's Knights Landing Platform as well as NVIDIA P100 GPUs, we demonstrate that it is possible to solve 3 times larger problems than the on-chip memory size with at most 15% loss in efficiency.

CCS CONCEPTS

•Computing methodologies →Massively parallel and high-performance simulations; •Theory of computation →*Parallel algorithms;* •Software and its engineering →*Domain specific languages;*

KEYWORDS

Stacked Memory, GPU, KNL, Unified Memory, Tiling, Performance

ACM Reference format:
István Z Reguly, Gihan R. Mudalige, and Michael B. Giles. 2017. Beyond 16GB: Out-of-Core Stencil Computations. In *Proceedings of Workshop on Memory Centric Programming for HPC, Denver, CO, USA, November 12–17, 2017 (MCHPC'17),* 10 pages.
DOI: 10.1145/3145617.3145619

MCHPC'17, Denver, CO, USA
 978-1-4503-5131-7/17/11...$15.00
DOI: 10.1145/3145617.3145619

1 INTRODUCTION

Today's accelerators offer unparalleled computational throughput, as well as high amounts of bandwidth to a limited amount of on-chip or on-board memory. The size of this fast memory has been a significant limiting factor in their adoption, as for most problem classes, it sets an upper bound for the problem sizes that can be solved on any single device. For larger problems, one had to either use multiple GPUs or fall back to the CPU, which usually has at least an order of magnitude larger memory.

Another significant limiting factor is the speed at which data can be uploaded to the accelerator memory. There is a great disparity between the bandwidth from a large memory to the accelerator memory (typically from CPU memory through PCI-e) and the bandwidth of the accelerator (up to 45×). This traditionally meant that all data was uploaded to the accelerator memory initially, and stayed resident for the entirety of the application - yielding the aforementioned size limitation.

In data streaming type applications, where a chunk of data is uploaded, processed, then downloaded, the workload (usually larger than GPU memory) is partitioned into small chunks, so it's possible to overlap copies in both directions and computations. To efficiently utilise accelerator bandwidth, this also means that any data uploaded has to be accessed about as many times as this ratio between upload bandwidth and accelerator bandwidth; otherwise performance will be limited by upload speed. To efficiently utilise the accelerator's computational resources, the ratio is even more extreme: for a P100 GPU one would need to carry out about 2500 floating point operations for every float variable uploaded (10 TFlops/s, 16 GB/s PCI-e BW, 4 bytes/float).

Going into the exascale era, most of the upcoming large supercomputers will be built with chips featuring on-chip high-bandwidth memory: Intel's Knight's Landing and later generations have at least 16GB MCDRAM, with bandwidths over 500 GB/s, and NVIDIA's P100 and later GPUs also feature 16GB with 720 GB/s and more. To tackle the issue with slow upload speeds, both have moved away from PCI-e: Intel's chips are stand-alone, have direct access to DDR4 memory (90GB/s on KNL), and the stacked memory can be used either as a separate memory space (flat mode) or as a large cache (cache mode). NVIDIA has introduced NVLink, connecting their GPUs to IBM CPUs and other GPUs, with 40 GB/s (in both directions), and allows oversubscribing GPU memory through Unified Memory - practically allowing

NUMA Distance for Heterogeneous Memory

Sean Williams
New Mexico Consortium
swilliams@newmexicoconsortium.org

Latchesar Ionkov
Los Alamos National Laboratory
lionkov@lanl.gov

Michael Lang
Los Alamos National Laboratory
mlang@lanl.gov

ABSTRACT

Experience with Intel Xeon Phi suggests that NUMA alone is inadequate for assignment of pages to devices in heterogeneous memory systems. We argue that this is because NUMA is based on a single distance metric between all domains (i.e., number of devices "in between" the domains), while relationships between heterogeneous domains can and should be characterized by multiple metrics (e.g., latency, bandwidth, capacity). We therefore propose elaborating the concept of NUMA distance to give better and more intuitive control of placement of pages, while retaining most of the simplicity of the NUMA abstraction. This can be based on minor modification of the Linux kernel, with the possibility for further development by hardware vendors.

CCS CONCEPTS

• **Software and its engineering** → *Memory management*; *Allocation / deallocation strategies*;

ACM Reference Format:
Sean Williams, Latchesar Ionkov, and Michael Lang. 2017. NUMA Distance for Heterogeneous Memory. In *Proceedings of MCHPC'17: Workshop on Memory Centric Programming for HPC (MCHPC'17).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3145617.3145620

1 INTRODUCTION

We are chiefly concerned with the application developers' experience when interacting with heterogeneous memory. Of course, this is a problem about the future, and the future is manifold, so really we will discuss one likely configuration of complex memory: heterogeneous NUMA, which is not coincidentally the approach used by Intel for exposure of the MCDRAM on Xeon Phi [4].

Nonuniform memory access (NUMA) is an abstraction commonly deployed for multisocket machines, in which each socket has an associated memory controller. Because of the relative distances of processors to memory controllers, *access* to memory is nonuniform, but the memory itself is assumed to be homogeneous. This introduces several subservient abstractions: the NUMA node or domain, which contains processing and memory elements that "go together," for example, a processor and its closest memory controller; the distance between any pair of NUMA nodes; and policies that use distances to decide placement of allocations onto nodes.

The traditional theory of distance is that, since memory devices themselves are homogeneous, NUMA can be treated as though it were a network, such that distance implies the number of hops or the request latency between a sender and a receiver. This also implies a simple strategy to get reasonable performance from a multisocket NUMA machine: pin a process to a single NUMA node, and allocate memory in such a way as to minimize distance. The latter part is, naturally, the default memory policy on Linux.

Another assumption behind this formulation of NUMA is that different memory devices are preferential for different processes. That is, in the multisocket machine with homogeneous memory devices, there is no particular trade-off at play: allocations should, in general, always stay within their node[1]. This is another factor in why a simple default policy works out well, because no optimization is needed, as there are no competing goals.

We see cracks in this formulation when memory devices are inhomogeneous. When a Xeon Phi is configured to expose its MCDRAM as an explicit NUMA node, its devices are assigned to nodes such that processors are paired with ordinary DRAM, and MCDRAM resides in one or more memory-only nodes. Distances are assigned such that local DRAM is closest, then all other DRAM nodes have a next closest and equal distance, then local MCDRAM is next closest, then other MCDRAM nodes have furthest and equal distance. This may seem strange, since MCDRAM is in some respects "better" than DRAM yet is considered further away, but it makes perfect sense on closer inspection: First, MCDRAM is only better in some respects, and second, it is a limited resource, so a more sensible default is for allocations *not* to reside in MCDRAM. This configuration turns MCDRAM into an "opt-in" device, and NUMA is used for implementation.

This is a very reasonable thing to do in the short term, but we see several long-term problems. First, this is unportable, as opting an allocation in to the MCDRAM requires knowing that this is the game being played, and also knowing the meaning of the NUMA node ID numbers. Second, with the existing memory policies, this works because there are only two nodes of interest (i.e., the local DRAM and local MCDRAM nodes). There is a memory policy to handle this situation, in which placement on an explicit, user-selected node is preferred, and if that cannot be accommodated, then placement falls back to the default policy.

In our view, the real culprit here is the implicit view that one distance metric adequately captures users' needs toward and beliefs about heterogeneous memory devices. As stated above, this view includes an assumption that memory devices are themselves homogeneous, and the strangeness surrounding Xeon Phi reflects

[1] There is another, less common use: if one is more concerned about bandwidth than latency, or one does not expect use of an allocation to be confined to one process or processor, then spreading an allocation across NUMA node can be useful. Fittingly, there is an interleave memory policy that handles this situation.

MCHPC'17, November 12–17, 2017, Denver, CO, USA

ACM ISBN 978-1-4503-5131-7/17/11…$15.00
https://doi.org/10.1145/3145617.3145620

Beyond 16GB: Out-of-Core Stencil Computations

István Z Reguly
Faculty of Information Technology and Bionics,Pázmány Péter Catholic University
Budapest, Hungary
reguly.istvan@itk.ppke.hu

Gihan R. Mudalige
Department of Computer Science, University of Warwick
Coventry, UK
g.mudalige@warwick.ac.uk

Michael B. Giles
Maths Institute, University of Oxford
Oxford, UK
mike.giles@maths.ox.ac.uk

ABSTRACT

Stencil computations are a key class of applications, widely used in the scientific computing community, and a class that has particularly benefited from performance improvements on architectures with high memory bandwidth. Unfortunately, such architectures come with a limited amount of fast memory, which is limiting the size of the problems that can be efficiently solved. In this paper, we address this challenge by applying the well-known cache-blocking tiling technique to large scale stencil codes implemented using the OPS domain specific language, such as CloverLeaf 2D, CloverLeaf 3D, and OpenSBLI. We introduce a number of techniques and optimisations to help manage data resident in fast memory, and minimise data movement. Evaluating our work on Intel's Knights Landing Platform as well as NVIDIA P100 GPUs, we demonstrate that it is possible to solve 3 times larger problems than the on-chip memory size with at most 15% loss in efficiency.

CCS CONCEPTS

•Computing methodologies →Massively parallel and high-performance simulations; •Theory of computation →*Parallel algorithms;* •Software and its engineering →*Domain specific languages;*

KEYWORDS

Stacked Memory, GPU, KNL, Unified Memory, Tiling, Performance

ACM Reference format:
István Z Reguly, Gihan R. Mudalige, and Michael B. Giles. 2017. Beyond 16GB: Out-of-Core Stencil Computations. In *Proceedings of Workshop on Memory Centric Programming for HPC, Denver, CO, USA, November 12–17, 2017 (MCHPC'17),* 10 pages.
DOI: 10.1145/3145617.3145619

MCHPC'17, Denver, CO, USA
 978-1-4503-5131-7/17/11...$15.00
DOI: 10.1145/3145617.3145619

1 INTRODUCTION

Today's accelerators offer unparalleled computational throughput, as well as high amounts of bandwidth to a limited amount of on-chip or on-board memory. The size of this fast memory has been a significant limiting factor in their adoption, as for most problem classes, it sets an upper bound for the problem sizes that can be solved on any single device. For larger problems, one had to either use multiple GPUs or fall back to the CPU, which usually has at least an order of magnitude larger memory.

Another significant limiting factor is the speed at which data can be uploaded to the accelerator memory. There is a great disparity between the bandwidth from a large memory to the accelerator memory (typically from CPU memory through PCI-e) and the bandwidth of the accelerator (up to 45×). This traditionally meant that all data was uploaded to the accelerator memory initially, and stayed resident for the entirety of the application - yielding the aforementioned size limitation.

In data streaming type applications, where a chunk of data is uploaded, processed, then downloaded, the workload (usually larger than GPU memory) is partitioned into small chunks, so it's possible to overlap copies in both directions and computations. To efficiently utilise accelerator bandwidth, this also means that any data uploaded has to be accessed about as many times as this ratio between upload bandwidth and accelerator bandwidth; otherwise performance will be limited by upload speed. To efficiently utilise the accelerator's computational resources, the ratio is even more extreme: for a P100 GPU one would need to carry out about 2500 floating point operations for every float variable uploaded (10 TFlops/s, 16 GB/s PCI-e BW, 4 bytes/float).

Going into the exascale era, most of the upcoming large supercomputers will be built with chips featuring on-chip high-bandwidth memory: Intel's Knight's Landing and later generations have at least 16GB MCDRAM, with bandwidths over 500 GB/s, and NVIDIA's P100 and later GPUs also feature 16GB with 720 GB/s and more. To tackle the issue with slow upload speeds, both have moved away from PCI-e: Intel's chips are stand-alone, have direct access to DDR4 memory (90GB/s on KNL), and the stacked memory can be used either as a separate memory space (flat mode) or as a large cache (cache mode). NVIDIA has introduced NVLink, connecting their GPUs to IBM CPUs and other GPUs, with 40 GB/s (in both directions), and allows oversubscribing GPU memory through Unified Memory - practically allowing