

Integrating Chapel programs and MPI-Based Libraries for High-performance Graph Analysis

Summer project:

Trevor McCrary (Mississippi State University, SNL intern)

Karen Devine, SNL

Andrew Younge, SNL



Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.



Chapel and Grafiki integration: using the right tool for every job



Chapel's PGAS enables analysts to easily manipulate parallel graph data (e.g., extract largest connected component via label propagation)



Grafiki's linear-algebra-based parallel graph algorithms enable efficient parallel graph analysis

(e.g., vertex hitting times)

Composing tools is challenging

- Different parallel paradigms: PGAS in Chapel vs MPI in Grafiki
- Different data formats: Edge lists in Chapel vs matrix in Grafiki
- Huge graphs prohibit copying/reformatting data in memory
- Three new approaches to integrate Chapel graph manipulation with Grafiki graph analysis *without requiring additional copy of graph*
 - Direct calls from Chapel to Grafiki using Chapel's C interface
 - Separate Chapel & Grafiki processes access shared mmap memory
 - Chapel and *Containerized* Grafiki access shared mmap memory



Requirement to not copy data allows ability to solve larger problems

- Share data with files is not scalable
 - Chapel reads data, modifies it, and writes results to a new file
 - C++ program reads new file and performs some analysis with it
- Sharing data with native data structures requires copies that use extra memory
 - Chapel reads data, modifies it, and shares it with C++ program
 - C++ program rearranges data, which creates a copy, and performs some analysis with it



C++ program needs to have general abstractions to use data in any format

- Grafiki relies on Trilinos for linear algebra classes and linear/eigen solvers
- Replaced Grafiki's use of concrete CrsMatrix class with abstract RowMatrix
 - RowMatrix user provides implementation of key operations (SpMV, norms, etc.)
- New Chapel-based RowMatrix uses edge lists directly from Chapel to perform matrix operations – *no data copy*



Approach 1: Chapel program calls Grafiki through Chapel's C interface



Chapel creates EdgeList in BlockDomain arrays Each Chapel locale directly calls C function (in *coforall*) with pointer to its local data C function instantiates RowMatrix from EdgeList pointers C function calls Grafiki

Grafiki computes using RowMatrix

Pro: Simple proof-of-concept for data sharing Con: Intrusive to user: needs to link with Trilinos and Grafiki



Approach 1 details: Coordinating Chapel locales and MPI ranks

- Numbers of Chapel locales and MPI ranks are equal
- Calls from Chapel to Grafiki must be made from each locale in parallel
 - All locales must enter Grafiki together to avoid hanging in Grafiki's MPI collective communication
 - Chapel coforall launches one task per locale
- Locales provide pointers to local edge list arrays

```
coforall loc in Locales {
    on loc {
        var subdom = edgelist.localSubdomain();
        call_C_function(c_ptrTo(edgelist[subdom.low]), ...);
    }
}
```



Approach 2: Separate Chapel and Grafiki processes share data in mapped memory



Chapel creates EdgeList in *new shareBlockDomain* arrays that

use mmap memory

Chapel program signals (via POSIX semaphore) external C process to begin

C process instantiates RowMatrix from EdgeList pointers in mmap'ed memory

C process calls Grafiki, signals Chapel when done

Grafiki computes using RowMatrix

Less intrusive: user uses Chapel as usual, only substituting shareBlockDomain for BlockDomain for shared data

6/15/22



Approach 2 details: new shareBlockDomain

- Chapel Domains
 - describe distribution of data across locales (processors)
 - manage global address space indexing
- Chapel's Block Domain assigns contiguous chunks of global arrays to locales
 - Each locale's local array is a separate allocation
- New shareBlock Domain replaces the local array allocation with mmap regions in shared memory
 - Separate mmap backing file for each shared array
 - Backing file names are shared between processes through tiny meta-data file
 - Chapel users simply substitute shareBlock for Block in their code





Approach 3: Chapel and Containerized Grafiki further simplify user experience



Least intrusive: Container handles details of building and running C process and Grafiki

6/15/22



Demonstrations done with Chapel on MPI and SMP systems

• Shared-memory Chapel on Xeon (kahuna)

- Single node, multiple locales / ranks per node
- Uses containerized Grafiki with Singularity
- MPI-enabled Chapel on Cray / Haswell (mutrino)
 - Single locale / rank per node, multiple nodes

• Compare

- approach One Chapel's C interoperability
- approach Two separate processes
- Look at Chapel label propagation and Grafiki hitting times



		Ch	apel	Grafiki	
		LabelPro	opagation	Hitting	g Times
	Number	Time pe	r iteration	Time per iteration	
	of	(sec	onds)	(seco	onds)
Platform	Locales	One	Two	One	Two
Kahuna	1	1.24	1.22	0.0285	0.0287
	2	NA 1.91		NA	0.0162
	4	NA 1.56		NA	0.0087
	8	NA 1.00		NA	0.0045
	16	NA	0.65	NA	0.0051
Mutrino	1	0.82	0.89	0.0215	0.0230
	2	0.65	0.77	0.0113	0.0120
	4	0.44	0.47	0.0058	0.0062
	8	0.22	0.29	0.0031	0.0033
	16	0.11	0.15	0.0018	0.0018

Small Matrix Bcsstk29.mtx 600k nonzeros 14k rows & columns



		Chapel			Grafiki			Kahuna used chapel
		LabelPropagation			Hitting Times			SMP 1.23, and we were
	Number	Time per iteration		ר	Time per iteration			unable to run approach
	of	(seconds)			(seconds)		s)	1 on more than one
Platform	Locales	One Two			One		Two	locale
Kahuna	1	1.24	1.22	0	.0285		0.0287	
	2	NA	1.91		NA		0.0162	
	4	NA	1.56		NA		0.0087	
	8	NA	1.00		NA		0.0045	
	16	NA	0.65		NA		0.0051	
Mutrino	1	0.82	0.89	0	.0215		0.0230	
	2	0.65	0.77	0	.0113		0.0120	Orea all Matrix
	4	0.44	0.47	0	.0058		0.0062	Small Matrix
	8	0.22	0.29	0	.0031		0.0033	BCSSIKZY.IIIIX
	16	0.11	0.15	0	.0018		0.0018	14k rows & columns



		Chapel		Gr	afiki	mmap memory
		LabelPropagation		Hitting Times		performs equal to
	Number	Time per iteration		Time per	r iteration	Chapel's usual memory
	of	(seconds)		(seconds)		when comparing
Platform	Locales	One	Two	One	Two	approach One and Two
Kahuna	1	1.24	1.22	0.0285	0.0287	
	2	NA	1.91	NA	0.0162	
	4	NA	1.56	NA	0.0087	
	8	NA	1.00	NA	0.0045	
	16	NA	0.65	NA	0.0051	
Mutrino	1	0.82	0.89	0.0215	0.0230	
	2	0.65	0.77	0.0113	0.0120	
	4	0.44	0.47	0.0058	0.0062	Small Matrix
	8	0.22	0.29	0.0031	0.0033	BCSSTK29.MTX
	16	0.11	0.15	0.0018	0.0018	14k rows & columns



		Chapel		Gı	rafiki	
		LabelPropagation		Hitting Times		
	Number	Time per iteration		Time per iteration		
	of	(seconds)		(seconds)		
Platform	Locales	One	Two	One	Two	
Kahuna	1	8626	7476	1187	1235	
	4	NA	9103	NA	364	
	16	NA	3749	NA	188	
Mutrino	4	OOM	10333	OOM	371	Large Matrix
	16	1274	3079	103	105	GAP_kron.mtx
	64	320	824	30	26	4B nonzeros
·	'1341/1 rows					



Kahuna used chapel SMP 1.23, and we were unable to run approach 1 on more than one locale

		Chapel		Gi	rafiki	
		LabelPr	opagation	Hitting Times		
	Number	Time per iteration		Time per iteration		
	of	(seconds)		(seconds)		
Platform	Locales	One	Two	One	Two	
Kahuna	1	8626	7476	1187	1235	
	4	NA	9103	NA	364	
	16	NA	3749	NA	188	
Mutrino	4	OOM	10333	OOM	371	Large Matrix
	16	1274	3079	103	105	GAP_kron.mt
	64	320	824	30	26	4B nonzeros
	101	520	024	50	20	134M rows



Using mmap memory shows loss of performance in Chapel algorithm but not in C++ algorithm

There is a noticeable loss of performance in the Chapel label propagation using mmap memory. We hypothesize that this is due to using Chapel 1.23's Block Domain as a template for share block while mutrino uses Chapel 1.24.

		Chapel		Gı	rafiki	
		LabelPropagation		Hittin	ıg Times	
	Number	Time per iteration		Time per iteration		
	of	(seconds)		(sec	conds)	
Platform	Locales	One	Two	One	Two	
Kahuna	1	8626	7476	1187	1235	
	4	NA	9103	NA	364	
	16	NA	3749	NA	188	
Mutrino	4	OOM	10333	OOM	371	Large Matrix
	16	1274	3079	103	105	GAP_kron.mtx
	64	320	824	30	26	4B nonzeros
Mutrino	4 16 4 16 64	NA NA OOM 1274 320	9103 3749 10333 3079 824	NA NA OOM 103 30	364 188 371 105 26	Large Matrix GAP_kron.m 4B nonzeros 134M rows



Using mmap memory shows loss of performance in Chapel algorithm but not in C++ algorithm

We still observe no performance penalty when using the mmap memory in the C++ program

		Chapel		Gr	afiki	
		LabelPropagation		Hitting Times		
	Number	Time per iteration		Time per iteration		
	of	(seconds)		(seconds)		
Platform	Locales	One	Two	One	Two	
Kahuna	1	8626	7476	1187	1235	
	4	NA	9103	NA	364	
	16	NA	3749	NA	188	
Mutrino	4	OOM	10333	OOM	371	Large Matrix
	16	1274	3079	103	105	GAP_kron.mtx
	64	320	824	30	26	4B nonzeros



Conclusion

- We present 3 new approaches for sharing data between Chapel and C++ without making multiple copies of data
 - Allows Chapel users to leverage existing fast
 C++/MPI based libraries



Future Work

- Extend operability to different Chapel distributions
 - Option for Chapel to accept a user defined allocator
- Analyze container performance
- Analyze Chapel and C++ integrated code against pure Chapel implementation