

# THE IMPACT OF MULTI-CORE ARCHITECTURES ON TASK RETRIEVAL POLICIES FOR VOLUNTEER COMPUTING

David Toth

Department of Computer Science, Merrimack College  
North Andover, MA 01845

U.S.A.

tothd@merrimack.edu

## ABSTRACT

Volunteer computing projects use donated CPU time to solve problems that would otherwise be too computationally intensive to solve. The donated CPU time comes from computers whose owners install a volunteer computing client program on their computer, allowing a project to use the computer's idle time. The low participation rate in volunteer computing and the increasing number of volunteer computing projects make improvements that more effectively use the donated CPU cycles very important. Past work showed that using certain task retrieval policies could increase the number of tasks volunteer computing clients complete. However, the past work assumed that the volunteered computers had a single CPU and the task retrieval methods that resulted in more completed tasks required the client to be connected to the Internet more often than the other policies. We simulated the task retrieval policies for computers with multi-core CPUs and found that in most cases, the multi-core architecture can lead to a slightly greater than linear increase in the number of tasks that the clients complete, relative to the number of cores the computer running the client has. Additionally, the multi-core architecture can reduce the performance gap between the best and worst performing policies significantly, affecting which policies are used.

## KEY WORDS

Volunteer computing, multi-core, distributed computing, performance

## 1. Introduction

Volunteer computing projects are used to advance knowledge in the fields of mathematics, the sciences, and medicine, as well as other disciplines. These projects allow people to make significant progress on problems that would otherwise be too computationally intensive to solve without the aid of a supercomputer. A volunteer computing project sponsor provides one or more servers with Internet access, data to analyze, and an algorithm to analyze the data. Volunteers install a client

program on their computers that usually runs as a screensaver or a background service/daemon process. When the client program runs, it retrieves data sets from the servers and analyzes the data using the algorithm provided by the volunteer computing project sponsor, returning the results when the analysis has been completed. The servers distribute the data to volunteered computers and aggregate the results returned by the clients. Because the client program running on the volunteered computers does almost all of the computing for the project and the volunteered computers can number in the millions, small performance improvements to the client can produce a significant performance improvement for a project.

As volunteer computing has matured over the years, it has become seen as a good way to solve computationally intensive problems related to topics that interest people. The development of the Berkeley Open Infrastructure for Network Computing (BOINC), framework allows people to develop volunteer computing projects much faster than they could before [1]. BOINC allows a project's sponsor to avoid writing the vast majority of the code involved in a volunteer computing project, so aside from the servers, the majority of what the sponsor needs to supply is the algorithm and the data to be analyzed [1]. The development of BOINC has likely been a key factor in enabling the number of volunteer computing projects to grow from the handful of the original and more well-known volunteer computing projects to the current much larger number. The original projects include SETI@home, Folding@home, and the Great Internet Mersenne Prime Search (GIMPS), while newer projects include Einstein@home and LHC@home [2, 3, 4, 5, 6].

Although the number of volunteer computing projects has increased dramatically over the years, the participation in volunteer computing has been reported to be less than 1% of the estimated 300 million Internet-connected computers [7]. Due to the computationally intensive nature of volunteer computing projects and the growth in the number of projects, attracting new participants and improving the effective use of donated CPU cycles are critical to the continued success of volunteer computing as a feasible paradigm for solving

large problems. Work by Toth and Finkel identified that the method that volunteer computing clients use to retrieve data sets to process could impact the number of tasks that volunteer computing clients completed by as much as almost 3% under certain conditions [8, 9]. However, that work assumed that computers had a single CPU. Additionally, the task retrieval methods that resulted in more completed tasks required the client to be connected to the Internet more often than the other policies. With the majority of computers being advertised at retailers like Best Buy and Circuit City having multi-core CPUs, it is important to examine the performance of the task retrieval policies for computers utilizing the multi-core architecture.

## 2. Methodology

We began by calculating the number of tasks that we would expect a computer with an n-core CPU to complete if each core worked on a separate task, based on the results from [8, 9]. This value gave us a baseline to compare the number of tasks a computer with an n-core CPU would complete if each core could work on the same task at once without incurring additional overhead and assuming that the task could be decomposed into n even sub-tasks that could be executed in parallel without additional cost. In order to provide a proper comparison

with the single CPU results from [8, 9], we started with the simulator used in [8, 9] and modified it to work for computers with multi-core CPUs. We use the same policies used in [8, 9] to maintain consistency with that work so we can compare our results. The policies we used are summarized in Table 1 and are *Buffer Multiple Tasks* (currently used by many volunteer computing projects including the projects built with the BOINC framework), *Buffer None* (currently used by some volunteer computing projects), *Buffer One Task* (a policy we devised), and *Download Early* (a policy we devised). The *Buffer Multiple Tasks* policy buffers as many tasks as the client could complete in a number of days specified by the user, based on the past availability of the client. The *Buffer None* policy does not buffer any tasks. When a client using the *Buffer None* policy completes a task it returns the result and then begins to download the next task to perform. The *Buffer One Task* policy keeps one task in its buffer. When the task being performed is completed, the task in the buffer is started and the client begins downloading another task which is stored in the buffer. The *Download Early* policy behaves like the *Buffer None* policy, except it begins downloading the next task when the task being performed reaches 95% complete.

We ran the simulations with the same traces that were used in [8, 9] so we could compare the results. The traces were collected in [10] and by sampling the status of

**Table 1 - Summary of Task Retrieval Policies**

	<b>Policy Name</b>	<b>Description</b>
1	Buffer Multiple Tasks	Buffers an amount of tasks that could potentially be completed in a number of days specified by the user.
2	Buffer One Task	Buffers one task. Every time the task in the buffer is removed and started, another task is downloaded.
3	Download Early	Buffers no tasks except when the current task being performed reaches 95% complete, the next task is downloaded.
4	Buffer None	Buffers no tasks. The next task is downloaded after the current task being performed is completed and the result is returned.

each computer's screensaver every 10 seconds, showed when each of 140 computers was on, on with the screensaver running, and off over a period of 28 days. The computers were home computers, business computers, computers belonging to undergraduate students at Worcester Polytechnic Institute, and computers in publicly accessible labs at Worcester Polytechnic Institute. The simulation parameters we used were the same as the ones used in [8, 9] to provide comparable results. The parameters were :

- File Size. The size of the file required by each task. The file is downloaded before the task can be started.
- Download Speed. The download speed of the client.
- Completion Time. The amount of CPU time required to complete a task if it is not interrupted.
- Buffered Days of Work. The number of days of work a client using the *Buffer Multiple Tasks* policy tries to buffer. This parameter is not relevant for the other task retrieval policies.
- Retrieval Method. The method clients use to retrieve tasks.
- Delay Bound. The amount of wall clock time between completing the download of a task and when the task will be aborted.
- Checkpoint Time. The time to create or restore a checkpoint.
- Internet Connectivity. We assume that the client is always connected to the Internet because we

do not expect mobile devices to be used to run volunteer computing clients [8].

The values used in the simulations were the ones determined in [8, 9] to be the most likely values to be encountered in a volunteer computing project. The values used were:

- **File Size:** 1 MB
- **Download Speed:** 300 kbps, 10 mbps
- **Delay Bound:** 7 days
- **Checkpoint Cost and Frequency:** 10 seconds and 1574 seconds
- **Task Duration:** 4 hours, 24 hours

### 3. Results & Analysis

We ran simulations for CPUs with 2, 3, 4, 8, and 16 cores. The results of our simulations are shown in Table 2, Table 3, Table 4, and Table 5. We found that in 138 of the 140 cases, the number of tasks completed exceeded the predicted number of tasks completed, resulting in a slightly better than linear increase in the number of tasks that were completed in relation to the number of processing cores used. This means that by having the processing cores in a computer's CPU all work on one task simultaneously, the number of tasks that the computer would complete would be more than if each core worked on a separate task. The two cases where the

number of completed tasks did not exceed the predicted number of completed tasks were when the download speed was slow (300 kbps) and the tasks were short (4 hours of CPU time was required for a single core to complete a task). We note that increasing the time required to complete a task or increasing the download speed once again resulted in every policy completing more tasks than predicted. In general, more tasks were completed than predicted because some tasks that were aborted when run on a single core CPU were completed because it took less wall clock time to complete them with multiple cores. Also, time that was spent on tasks that were aborted when run on a single core CPU in some cases was no longer wasted when run on multiple cores and could be used to complete additional tasks. We observed that in general, the amount by which the expected number of completed tasks was exceeded increased as the number of processing cores used increased and increased more for longer tasks. Our simulations suggest that for the 24 hour tasks, a computer with a quad-core CPU could increase the number of tasks it completed by almost 4% for some task retrieval policies and as much as 6.5% for other policies if the cores worked collaboratively on one task at a time rather than working on separate tasks. We also observed that the policies that cause a client to buffer tasks exceed the predicted number of completed tasks by a higher percentage than the policies that do not buffer tasks. Increasing the amount of tasks buffered resulted in exceeding the predicted number of completed tasks by a

**Table 2 - Simulation Results for Download Speed 300 kbps and 4 Hour Tasks**

		Task Retrieval Policy						
		Buffer None	Download Early	Buffer 1 Task	Buffer 1 Day	Buffer 3.5 Days	Buffer 7 Days	Buffer 14 Days
Dual Core	Multiple of Tasks Completed by Single Core CPU	2.00	2.01	2.01	2.01	2.01	2.02	2.02
	% Improvement Over Predicted Value	0.22	0.39	0.53	0.52	0.68	1.11	0.84
Triple Core	Multiple of Tasks Completed by Single Core CPU	3.00	3.01	3.02	3.02	3.02	3.04	3.03
	% Improvement Over Predicted Value	0.05	0.44	0.59	0.60	0.79	1.25	1.10
Quad Core	Multiple of Tasks Completed by Single Core CPU	4.00	4.03	4.03	4.03	4.04	4.06	4.06
	% Improvement Over Predicted Value	0.03	0.64	0.80	0.84	1.02	1.49	1.42
8 Core	Multiple of Tasks Completed by Single Core CPU	7.95	8.06	8.07	8.08	8.09	8.13	8.12
	% Improvement Over Predicted Value	-0.63	0.74	0.89	0.94	1.12	1.57	1.52
16 Core	Multiple of Tasks Completed by Single Core CPU	15.79	16.25	16.28	16.29	16.31	16.38	16.37
	% Improvement Over Predicted Value	-1.28	1.59	1.75	1.79	1.95	2.39	2.29

**Table 3 - Simulation Results for Download Speed 10 mbps and 4 Hour Tasks**

		Task Retrieval Policy						
		Buffer None	Download Early	Buffer 1 Task	Buffer 1 Day	Buffer 3.5 Days	Buffer 7 Days	Buffer 14 Days
Dual Core	Multiple of Tasks Completed by Single Core CPU	2.01	2.01	2.01	2.01	2.01	2.02	2.02
	% Improvement Over Predicted Value	0.37	0.39	0.53	0.53	0.68	1.12	0.80
Triple Core	Multiple of Tasks Completed by Single Core CPU	3.01	3.01	3.02	3.02	3.02	3.04	3.03
	% Improvement Over Predicted Value	0.41	0.44	0.59	0.61	0.78	1.25	1.13
Quad Core	Multiple of Tasks Completed by Single Core CPU	4.02	4.03	4.03	4.03	4.04	4.06	4.06
	% Improvement Over Predicted Value	0.62	0.65	0.80	0.85	1.02	1.52	1.49
8 Core	Multiple of Tasks Completed by Single Core CPU	8.05	8.06	8.07	8.08	8.09	8.13	8.13
	% Improvement Over Predicted Value	0.67	0.74	0.89	0.95	1.14	1.64	1.63
16 Core	Multiple of Tasks Completed by Single Core CPU	16.24	16.26	16.28	16.29	16.32	16.40	16.40
	% Improvement Over Predicted Value	1.48	1.59	1.75	1.81	2.00	2.51	2.50

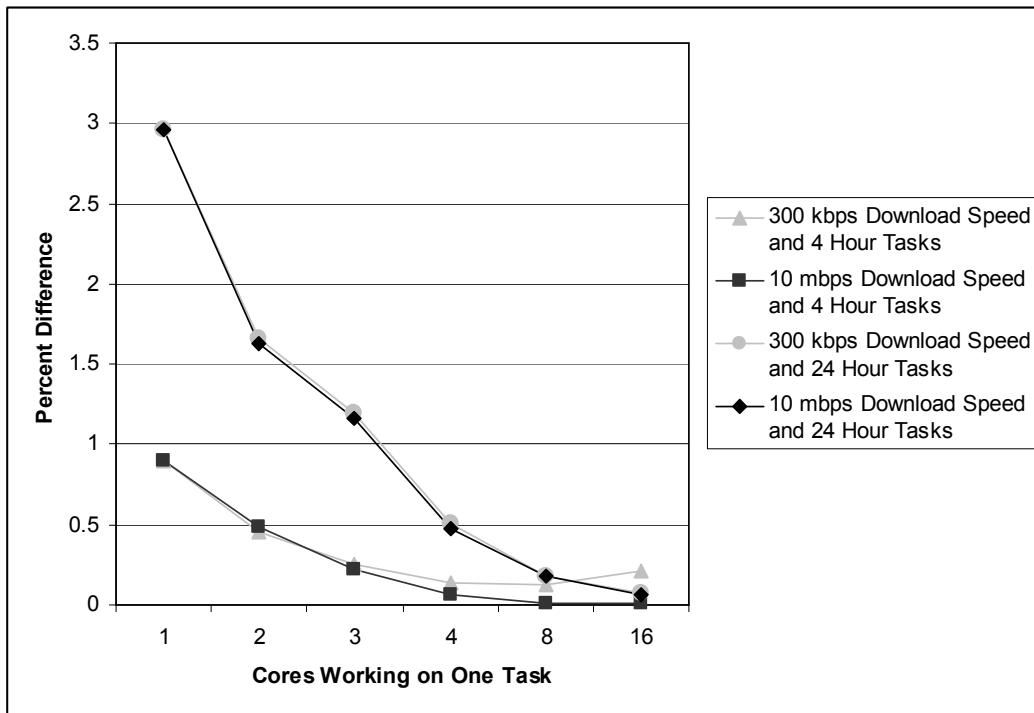
**Table 4 - Simulation Results for Download Speed 300 kbps and 24 Hour Tasks**

		Task Retrieval Policy						
		Buffer None	Download Early	Buffer 1 Task	Buffer 1 Day	Buffer 3.5 Days	Buffer 7 Days	Buffer 14 Days
Dual Core	Multiple of Tasks Completed by Single Core CPU	2.05	2.05	2.05	2.06	2.09	2.10	2.07
	% Improvement Over Predicted Value	2.32	2.37	2.60	3.04	4.40	4.80	3.75
Triple Core	Multiple of Tasks Completed by Single Core CPU	3.10	3.11	3.11	3.12	3.16	3.18	3.16
	% Improvement Over Predicted Value	3.39	3.50	3.79	4.00	5.43	6.10	5.38
Quad Core	Multiple of Tasks Completed by Single Core CPU	4.15	4.16	4.17	4.18	4.24	4.27	4.26
	% Improvement Over Predicted Value	3.84	3.90	4.15	4.45	5.92	6.73	6.53
8 Core	Multiple of Tasks Completed by Single Core CPU	8.34	8.36	8.40	8.42	8.54	8.61	8.60
	% Improvement Over Predicted Value	4.31	4.54	5.01	5.30	6.72	7.58	7.54
16 Core	Multiple of Tasks Completed by Single Core CPU	16.70	16.77	16.85	16.90	17.14	17.27	17.27
	% Improvement Over Predicted Value	4.34	4.83	5.33	5.61	7.10	7.96	7.95

**Table 5 - Simulation Results for Download Speed 10 mbps and 24 Hour Tasks**

		Task Retrieval Policy						
		Buffer None	Download Early	Buffer 1 Task	Buffer 1 Day	Buffer 3.5 Days	Buffer 7 Days	Buffer 14 Days
Dual Core	Multiple of Tasks Completed by Single Core CPU	2.05	2.05	2.05	2.06	2.09	2.10	2.08
	% Improvement Over Predicted Value	2.37	2.37	2.60	3.04	4.40	4.83	3.77
Triple Core	Multiple of Tasks Completed by Single Core CPU	3.11	3.11	3.11	3.12	3.16	3.18	3.16
	% Improvement Over Predicted Value	3.50	3.50	3.79	4.00	5.43	6.10	5.42
Quad Core	Multiple of Tasks Completed by Single Core CPU	4.16	4.16	4.17	4.18	4.24	4.27	4.26
	% Improvement Over Predicted Value	3.90	3.90	4.15	4.45	5.92	6.73	6.56
8 Core	Multiple of Tasks Completed by Single Core CPU	8.36	8.36	8.40	8.42	8.54	8.61	8.60
	% Improvement Over Predicted Value	4.54	4.54	5.01	5.30	6.72	7.57	7.54
16 Core	Multiple of Tasks Completed by Single Core CPU	16.77	16.77	16.85	16.90	17.14	17.28	17.27
	% Improvement Over Predicted Value	4.81	4.83	5.33	5.61	7.11	7.97	7.96

**Figure 1 – Difference Between Number of Tasks Completed by the Policies that Complete the Most and Fewest Tasks**



larger amount, except for the policy that buffered 14 days of tasks. The policy that buffered 14 days of tasks had a percent improvement in completed tasks that was slightly lower than the policy that buffered 7 days of tasks. The

fact that policies that buffered tasks had larger increases in the percent improvement over the predicted number of completed tasks than the policies that did not buffer tasks is significant. This means that the gap between the

numbers of tasks that are completed by clients using different policies shrinks as the number of CPU cores used to work on a single task increases. The shrinking of that gap means that using task retrieval policies that buffer work will not result in as significant a decrease in the number of tasks completed as more cores work on a single task. In fact, in some cases, computers using multiple cores to work on a single task complete more tasks if they buffer 14 days of tasks than if they do not buffer any tasks which was not the case for a single core CPU. However, the *Download Early* policy still outperforms the *Buffer Multiple Tasks* policy in all cases. The shrinking of the gap allows us to relax the assumption from [8, 9] that computers are constantly connected to the Internet by using a policy that buffers tasks, while not sacrificing as much in the number of tasks that will be completed. Figure 1 shows the gap between the policy that completes the most tasks and the policy that completes the fewest tasks.

#### 4. Conclusion

The results of our simulations have shown two important conclusions. The results show that the multi-core architecture can lead to a slightly greater than linear increase in the number of tasks that the clients complete, relative to the number of cores the computer running the client has if the cores can work on a task simultaneously. This suggests that problems solved by volunteer computing projects should be formulated such that they can be decomposed into  $n$  equal parts that can be solved simultaneously if possible. In this case, a computer can complete more tasks than if each core worked on a separate task. Additionally, the multi-core architecture can reduce the performance gap between the best and worst performing task retrieval policies significantly. The reduction of the performance gap shows that policies that buffer tasks can perform close to as well as other policies for multi-core computers under the specified conditions. That will allow volunteer computing projects to code their client programs to use policies that buffer tasks, thus relaxing the assumption that computers must have a constant Internet connection, without losing a significant amount of performance from the volunteered computers and in some cases getting better performance.

Our results did make two assumptions beyond those made in [8, 9] that should be addressed in future work. We assumed that tasks could be decomposed into  $n$ -equal length sub-tasks without any overhead. We also assumed that completion of the tasks would not be slowed down by any additional contention for resources such as accessing the disk or communication busses. We intend to address those assumptions in future work. We also intend to study the effect of the different behaviour that volunteers may exhibit in an attempt to be more environmentally conscious. We recognize that some people who participate in volunteer computing will continue to participate by running the volunteer

computing client whenever their screensaver runs or in the background constantly while their computer is running. However, other people may choose to run a volunteer computing program only when their computer is actively in use and have it hibernate or put it in some other power-saving mode when they are not using it. In addition to this, with the multi-core computers that have become commonplace today, some people may choose to run the volunteer computing client only on some subset of the processing cores in their computer either all the time or while they are using the computer and it is not in the power-save mode.

#### References

- [1] David P. Anderson, BOINC: A System for Public-Resource Computing and Storage. *Proc. 5th IEEE/ACM International Workshop on Grid Computing*. Pittsburgh, PA, U.S.A. November 8, 2004.
- [2] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, & D. Werthimer, SETI@home: An Experiment in Public-Resource Computing, *Communications of the ACM*, 45(11), 2002, 56-61.
- [3] Folding@Home. "Folding@Home Distributed Computing." <http://folding.stanford.edu/> Updated 2006. Accessed 10/26/06.
- [4] GIMPS. "Mersenne Prime Search". <http://www.mersenne.org/prime.htm> Accessed 7/13/05.
- [5] Einstein@Home. <http://einstein.phys.uwm.edu/> Updated 8/14/06. Accessed 10/26/06.
- [6] LHC@Home. <http://lhathome.cern.ch/> Updated 9/29/06. Accessed 10/26/06.
- [7] J. Bohannon, Grassroots Supercomputing, *Science* 308, 2005, 810-813.
- [8] D. Toth and D. Finkel, Increasing the Amount of Work Completed by Volunteer Computing Projects with Task Distribution Policies, David Toth and David Finkel, *Proc. 2nd Workshop on Desktop Grids and Volunteer Computing Systems - PCGrid 2008*, April 18, 2008, Miami, FL, U.S.A.
- [9] D. Toth. Improving the Productivity of Volunteer Computing. Ph.D. Dissertation, May 2008.
- [10] D. Toth and D. Finkel, Characterizing Resource Availability for Volunteer Computing and its Impact on Task Distribution Methods, *Proc. 6th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems - SEPADS 2007*, Corfu, Greece, February 16-19, 2007.