Shared Data or Message-Passing - A Human Factor in Technical Choices?

A.C. Sodan
Computer Science
University of Windsor
Windsor, ON N9B 3P4, Canada
acsodan@cs.uwindsor.ca

L.F. Capretz
Electrical and Computer Engineering
University of Western Ontario
London, ON N6G 1H1, Canada
lcapretz@engga.uwo.ca

Abstract

It is an ongoing debate in parallel processing whether shared- or distributed-memory computing models are better, whether shared-data or message passing is preferable. Recent research has shed some more light on the debate, showing that many applications can be supported well in either model (though potentially with some special tuning for the corresponding machine) but that for some applications with more extreme behavior the corresponding machine type and computing model are preferable or even the only feasible solution. Otherwise, what does make people choose one or the other? We investigate the human factor and propose the model that the personality type determines to a large extent personal preferences. The paper discusses the relationship between certain personality types and the programming model. To determine these aspects, we applied the psychological Myer-Briggs-Type-Indicator test on a group of students for whom both programming models were mostly new (i.e. who were not pre-occupied). The results give reasonable evidence for the validity of our proposed model and the relevance of the human factor in technical choices, i.e. that choices are not only/always a matter of which model is "better".

Keywords: MPI, Multithreading, Personality Type, MBTI, Programming Model, Personal Preferences

1 The Parallel-Programming Model Debate

The most appropriate programming model for parallel processing is a topic of ongoing discussion. The High-Performance-Computing community is basically split in favoring either shared- or distributed-memory machines, a shared-data or message-passing programming model - if neglecting attempts to design a higher-level model which potentially hides the details of either approach and unifies them. The currently most important representatives are OpenMP [16] for shared-data programming and MPI [13] for message passing (we are not discussing the extension MPI-2 here), both being reasonable abstract and portable approaches to mostly synchronous data-parallel programming. MPI is clearly dominating in practical use. The basic model supported in both is SPMD, meaning that the same program is applied to different data sets. OpenMP supports loop-oriented expression of parallelism and synchronization on shared data. A more basic and more flexible approach is Pthreads [10], a library supporting the dynamic creation of individual threads and their synchronization and allowing the expression of more arbitrary and asynchronous parallelism. The individual tasks may be heterogeneous and the approach thus is meaningful for software modularization and I/O-latency hiding on uniprocessor machines. MPI supports one-way two-sided message-based communication via copying between multiple processes with separate address and data spaces and explicit send/receive. The processes are set up at program startup, i.e. the number is static. Messages may be either point-to-point or collective communication. MPI basically supports homogeneous parallelism. MPI also includes features to enable a more arbitrary style of interaction and dispatch between different heterogeneous activities. And Pthreads can be used to formulate completely synchronous programs. For a summary of Pthreads and MPI features, see Table 1.

Table 1: The different basic characteristics of shared-data and message passing via Pthreads and MPI. The approaches have also different characteristics with respect to scheduling and load balancing but these are not relevant for our case study.

Multithreading / Pthreads	SPMD / MPI
dynamic creation of individual threads	static number of processes
shared address/data space	separate data spaces via multiple processes
individual explicit synchronization	implicit synchronization via messages
data exchange via shared data	data exchange via messages
task parallelism	data parallelism
heterogeneous work possible	homogeneous work
asynchronous execution possible	mostly synchronous execution

Java - which is becoming increasingly important as a parallel language, too - both threads and message passing (via sockets or Remote Method Invocation) is supported. Socket communication is one-way like MPI. However, there is nothing such as an SPMD model but processes are individually set up and potentially communicate with each other.

Shared-data programming is perceived by many people as the easier variant, because it does not require to think about data ordering / layouts and about data exchange for semantic correctness - and no understanding of structural, e.g. geometric patterns. Sharing data and memory space requires a complete understanding of data accesses with respect to synchronization only. However, this is not the whole story when performance comes into play. The only thing that really can be saved is data exchange. Data ordering (and potentially reordering if the access pattern or the data structure changes) is necessary to obtain good cache locality on SMP and DSM machines [6,7] (which, however, is due to the fact that the shared memory in the end is not really homogeneous and a hierarchy). Indeed, the study in [18] on a dynamic, irregular N-body problem (including dynamic remeshing) program showed that, on a DSM machine (Origin2000), the shared-memory implementation was simpler, i.e., had significantly less lines of code (with proper coding, similar performance was obtained). However, synchronization being necessary can easily be overlooked and debugging be hard. The shared-data model naturally fits to machines with shared memory and exploits the performance benefits of these machines.

Message passing is the natural model on distributed-memory machines such as clusters, expressing that data being exchanged is to be send from memory to memory. However, message passing requires to carefully think about proper data ordering/ partitioning to exploit locality in memory and minimize communication cost. Furthermore, it requires thinking about which data has to be sent when and to organize the exchange. However, it requires hardly any additional synchronization and potential mistakes in the data exchange usually become obvious pretty quickly. Because of the separated address spaces, message passing is the safer way to program. Since distributed-memory machines scale better, message passing is the only feasible model for massively parallel applications.

The different arguments in favor of either model have led to supporting shared-data-style programming on distributed memory via VSM or DSM and message passing on shared memory - with the additional advantage of making programs portable between different types of machines. However, even the hardware-supported DSM has shown to require support by proper algorithms and proper data ordering [6,7] and the same applies to VSM. VSM was first perceived as a very promising solution to create a uniform programming approach for all machine types [9], but has shown not to be efficient enough unless there is hardly any sharing and accesses are pretty localized on certain pages per processor. In other words, without good locality, VSM can perform poorly. Conversely, message passing on shared memory can be supported by internal optimization to share data buffers and reduce copying cost [23]. Still there are applications with very tight data coupling, i.e. frequent or high-volume data exchange (even nearest neighbor can become a problem if too intensive) which perform better or are feasible only with explicitly sharing data and avoiding any copying. Despite these cases favoring either approach for clear technical reasons, there is a wide range of programs running well on either machine type and of cases where the transfer of one model to the non-corresponding machine type works well [15,23]. Furthermore, often different algorithms (more suitable for one or the other machine type) exist - which extends the range of choices. And actually the perception which programming model really is the easier one differs. Furthermore, currently we experience a more or less friendly co-existence of different machine types and programming models and partially a hybrid integration - such as SMP/MPP machines and corresponding OpenMP/MPI combinations [3]. A more detailed discussion of the characteristics and merits of the two approaches and their integration, can, e.g., be found in [21].

2 The Human Factor and MBTI as a Way to Measure It

According to the above discussion, there is clearly some room for personal preference. Moreover, technical discussion appears to some extent be often driven by the underlying personal preference. Though, the human factor is usually overlooked in parallel processing (with few exceptions like [17] investigating the human factor in, e.g. tool development), the discussion above pops up the question how the preferences can be explained and whether there is some correlation between the programming model and the personality types. In [20], we have performed a classification of personality types and their different strengths based on dual characterizations inspired from brain research [22]. MBTI (Myer-Briggs Type Indicator) as discussed below confines the classification to four pairs, based on Jung's theory of psychological types [8]. This provides a scheme that is easy enough to handle and sufficiently complex to perform non-trivial characterizations. Thus, we applied MBTI to characterize personality types in software engineering [5]. Furthermore, MBTI provides questionnaires allowing to rate people's preference in each pair and thus supporting empirical tests to link both personality type and computer-science approaches in a provable manner. Evaluations of types consider preferences in individual pairs and combinations of them in typically sets of 2 or all 4 dimensions, the latter constituting the so-called MBTI types. The latter options add the interesting approach to consider patterns, expressing colored/modified or enhanced personality characteristics in addition to the mere linear discussion of individual dimensions.

2.1 MBTI Types

Within the field of computer programming, there is agreement that there are tremendous differences among individuals' performance. Teachers of computer programming witness first hand the huge variety among students in learning achievement and programming assignments. In [19], it is reported that researchers have found differences as high as 10 to 1 of programmers (with similar background) regarding programming performance, and in [2], it is stated that one of the problems in studying introductory programming classes is the tremendous variability in achievement. Cognitive styles have been investigated as factors that may help explain some of the variability; however, they have failed to consistently explain individual differences in achievement. MBTI offers the potential to provide a suitable model.

MBTI is an instrument designed to measure four dimensions (dichotomies, consisting of two poles) of an individual's personality [12]. Each of the poles represents opposite preferences, and for every pair of items an individual prefers one pole over the other. MBTI's four internal dimensions relate to characteristic or preferred ways of becoming aware, reaching conclusions, making decisions, and being oriented either to a private inner world or external world of actions. These four dimensions are called sensing(S)-intuition(N), thinking(T)-feeling(F), perception(P)-judgement(J) and introversion(I)-extroversion(E), respectively.

Introversion/extraversion refers to a person's orientation toward the world. Extraversion describes an attitude where attention is drawn out towards objects and people. Introversion describes an attitude where attention is drawn toward the inner world of ideas. The second category (sensing/intuitive) refers to ways in which a person perceives information. A sensing person tends to perceive observable facts through the five senses. An intuitive person perceives information based on the meaning, relationships, and possibilities beyond the information gathered from the senses.

The thinking/feeling dimension relates to ways in which a person makes decisions. Individuals with high scores on the feeling pole make decisions based on their own feelings and the feelings of others. Those on the opposite end (thinking) base their decisions on objective, impersonal, and logical analysis of a situation. The last dimension refers to a person's orientation toward life. Judgers prefer to work in a linear, orderly manner, whereas, perceptives would rather live a flexible, spontaneous life.

To sum up, MBTI provides a measure of personality by looking at the various ways we prefer to receive information and perceive our surroundings. It describes preferences, it does not measure skills or abilities. Of course, people can and do use all eight preferences. In each of the four pairs, however, we all have one preference that is stronger than the other, one that works better for us than its complement.

Table 2: The 16 MBTI types and their distribution (and the concrete number in each class in parenthesis) among the general population (G) [12] and among 68 computer science students in Brazil (CSB) [5] and 31 computer science students at University of Windsor, Canada (CSW).

ISTJ
G=11.6%
CSB=19.1% (13)
CSW=25.8% (8)
	ISFJ
	G=13.8%
	CSB=3.0% (2)
	CSW=3.2% (1)
		INFJ
		G=1.5%
		CSB=1.5%(1)
		CSW=0% (0)
			INTJ
			G=2.1%
			CSB=7.3% (5)
			CSW=9.7% (3)

ISTP
G=5.4%
CSB=4.4% (3)
CSW=9.7% (3)
	ISFP
	G=8.8%
	CSB=4.4% (3)
	CSW=0% (0)
		INFP
		G=4.4%
		CSB=3.0% (2)
		CSW=0% (0)
			INTP
			G=3.3%
			CSB=13.2% (9)
			CSW=3.2% (1)

ESTP
G=4.3%
CSB=11.7% (8)
CSW=3.2% (1)
	ESFP
	G=8.5%
	CSB=1.5% (1)
	CSW=0% (0)
		ENFP
		G=8.1%
		CSB=3.0% (2)
		CSW=9.7% (3)
			ENTP
			G=3.2%
			CSB=7.3% (5)
			CSW=3.2% (1)

ESTJ
G=8.7%
CSB=11.7% (8)
CSW=16.2% (5)
	ESFJ
	G=12.3%
	CSB=3.0% (2)
	CSW=3.2% (1)
		ENFJ
		G=2.5%
		CSB=1.5%(1)
		CSW=0% (0)
			ENTJ
			G=1.8%
			CSB=4.4% (3)
			CSW=12.9%(4)

Table 2 shows the 16 types resulting from the different combinations of variables together with their distribution. We find a cluster of TJs in our sample. Especially, we can see the distribution for computer-science students to be even more biased toward TJ (20 out of 31 Windsor students). I.e. 64% fall into this category in comparison to 24.2 % of the general population (statistical significance of bias both vs. equal distribution and distribution in general polulation is p value < 0.001). TJs experience a compulsion to identify principles early so that they can be applied in an organized, orderly fashion, pushing for closure (judging). Individuals with the combination TJ are described as realistic, tough-minded people with a greater aptitude for technical as opposed to interpersonal skills. Thus, TJs appear to have the special skills required for most computer-science tasks This corresponds to results in [4] showing that certain types are indeed superior to others in accuracy of problem solving and decision making. Excep to TJ, the sample sizes are too small to draw further detailed conclusions. Note that MBTI says that all types are valuable and necessary, but each with its own special gifts and strengths. Thus, many NFs and SFs may be drawn to fields like psychology and school teaching because of their concern for others or find their niche in the more people-oriented aspects of software development.

2.2 Relating MBTI and MPI/Pthreads

The consistency of positive relationship among different types clearly reveals that tasks involved in programming have been associated with certain MBTI variables (dimension poles) (e.g., in [11]). For instance, Es prefer to work interactively with a succession of people, whereas Is prefer work that permit some solitude for concentration. Ss like details more than Ns; additionally, Ns like creativity, working in a succession of new problems and feeling more satisfied with complex problems than Ss. Ts prefer to analyze logical and objective data and, e.g., perform quantitative evaluations; Ts want work that requires logical thinking, whereas Fs want work that provides service to people. Js prefer work that imposes a need for order, whereas Ps prefer work that requires adapting to changing situations. This investigation provides empirical results to reach conclusions regarding the MBTI types and preferences related to programming.

The findings that Ss are twice as accurate than Ns in solving problems supports the expectations that Ss would be better in work requiring detail and precision. Ns prefer to pay more attention to the meaning behind the facts rather than to the facts themselves. ST and NT are similar to each other regarding accuracy of problem solving. Ns and Js took less time than their opposites to solve problems. Similarly, NTs took the least amount of time, while STs were less good as far as efficiency of problem solving is concerned. However, because being more precise, STs are expected to dominate in the field of parallel programming which is much about careful algorithm design and performance considerations.

STs like activities that require the use of well-learned knowledge, not the development of new solutions. They are very good observers and like details. NTs are creative and enjoy abstract symbolic relations, finding patterns rather than dealing with details. They like to create new knowledge rather than applying or fixing existing techniques. The NTs are more creative than STs because Ns see possibilities beyond the given facts, and look for patterns and relationships. They are more adept at identifying underlying principles than at memorizing specific data. Thus, NTs couple a theoretical framework and the tendency to extrapolate beyond the details, so that new principles can be seen.

Therefore, it is expected that STs would prefer a logical and scheduled approach as used in MPI, whereas NTs would prefer choosing Pthreads because it supports and demands more creativity and innovation. Furthermore, NTs are better in dealing with complexity. In addition, stronger boundaries between the conscious and the unconscious as an indicator for separateness were found for Ss, Ts, and Js [1]. This corresponds to the separation of data spaces performed in MPI. Correspondly, N, F, P are stronger in relatedness. For a summary of the expected preferences of certain MBTI types for either programming model, see Table 3.

Table 3: The different skill-requirements of shared-data and message passing via Pthreads and MPI and the corresponding MBTI variables.

Multithreading	MBTI relation	Message Passing	MBTI relation
requires understanding of data accesses with respect to principle sharing and data exchange only	N is more interested in patterns and less good in details	requires an understanding of data accesses with respect to data exchange, ordering and distribution and thus a more geometric thinking	S is more accurate than N
supports the formulation of algorithms with stronger relatedness	N,F,P apply a view with stronger relatedness	supports the formulation of programs with maximum separation of concerns	S,T,J are stronger in separation
allows programmers the formulation of more arbitrary, dynamic, irregular, heterogeneous and asynchronous programs, i.e. more complex control flows	Ns are more biased toward non-standard and innovative solutions, NTs can better deal with complexity	MPI: provides programmers a simple synchronous model with mostly uniform and statically determined control flow and fixed procedures/rules	Ss are more oriented toward fixed and standard guidelines, STs like the conventional
debugging requires dealing with complexity of control flow and a more creative approach	NTs can better deal with complexity and more easily invent specific testing procedures	safe and relatively easy to debug (especially if a mostly synchronous SPMD model) according to fixed procedures/rules	STs are more oriented toward fixed procedures/rules

3 Experimental Test Setting and Results

We have performed two tests. One on a group of 4th-year Honors and Graduate students (in a course on distributed and parallel programming), using Pthreads and MPI (referred to below as Test 1). The other one on a group of 3rd-year students (in a course on operating system principles), using Java threads and socket (Test 2). Most of the students met parallel programming models the first time. For the first group, the course was not a mandatory but elective one, i.e. we can assume that students having chosen the course are interested in this kind of programming. In their final project, they had to program a simple numeric grid problem (as used to model heat transfer on a metal sheet, for details see, e.g., [24]) in two different versions, one with MPI and one

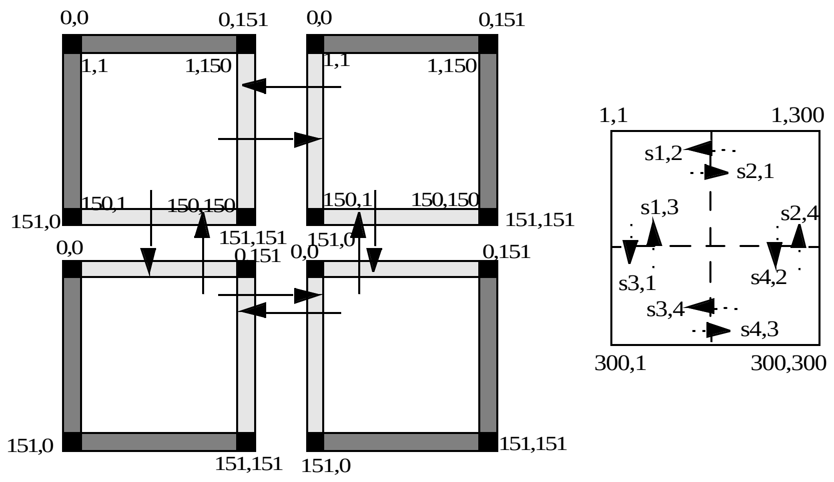

FIGURE 1. Data partitioning and organization of a 300x300 grid with ghostcells (left) using 4 MPI processes with equally structured subgrids. Arrows indicate the data exchange to be performed between iteration steps. The dark grey cells at the outside border represent initial values not being changed during the processing. Note that with the organization shown, the structure is very uniform, only requiring extra checks for each side whether there is a neighbor with whom to exchange data. The shared-memory version (small graphic right) uses a single 300x300 grid without any ghostcells but semaphores between neighbors to synchronize access to neighboring cells. s indicates semaphores.

with Pthreads. The problem is described below in more detail. 10 students participated voluntarily in the questionnaire which included the basic MBTI test (form G) and additional questions related to the numeric-grid project. The second test was about using a simple program with two to three Java threads for I/O-latency hiding and parallel processing and a simple program with Java sockets for client-server interaction. 21 students participated in this test voluntarily (out of 195 students in the course) for a small payment, acknowledging the time which they invested for this test. The participating students can be considered to be the more serious students with some understanding of science. The distribution of all the 31 participating students among the MBTI classes is shown in Table 2 for completeness.

Test 1: For the numeric grid, we work with a simple regular and rectangular grid, solving an equational system via iterative approximation. The application partitions well, i.e. processors can work mostly on their own data. However, the neighboring cells are shared. On distributed memory this means that the neighboring cells need to be exchanged between iteration steps and that so-called ghostcells are necessary to keep the neighbors' last-step data (see Figure 1). On shared memory, neither data copying nor ghostcells are necessary, but explicit synchronization is required on the availability of the neighboring-cell data to perform the next iteration step - which with message passing is performed implicitly. However, the MPI version works with separate uniform subgrids and the Pthreads version with one global grid. Both variants scale well because there is only nearest-neighbor data exchange. Absolute runtime is still reasonable when using MPI, especially if making the grid-size larger and more realistic than our example. Furthermore, worth to note is that the basic problem structure in both variants is similar, i.e. regular and static. The full capacity of Pthreads to formulize arbitrary parallelism is thus not exploited. Still there are significant differences. Furthermore, it is important to note that this application does not involve much potential for creative design, but mainly requires accuracy in formulation.

Test 2 The programs for the second test are fairly simple. The thread program uses one thread to read input data and another one to consume it. Data exchange is via a bounded buffer. This program requires an understanding of data sharing, synchronization, thread execution, and the indeterministic execution order. The socket program requires to setup a socket connection and to communicate between a client and a server to let the server perform certain tasks for the client. The thread assignment was a slightly more difficult and took the students more time to implement. Conversely to Test 1, both programs are asynchronous.

Both groups of students were asked

- whether they preferred the variant using messages or threads (1)
- which problems they encountered when they formulated programs in either style (2)

The detailed raw results are shown in Table 5, differentiated per MBTI type and per the two tests. Note that the sample size per type is too small to be of statistical significance. We therefore group into larger classes below (each grouping uses the whole sample set and divides into only two subclasses).

Table 4 shows the evaluation along pairs of MBTI variables. Overall, we can conclude that the majority of students (19 vs. 12) prefers threads though the corresponding assignment in Test 2 costed them more time to complete. This is a bias with a clear significance (p value is < 0.035). As is stressed by the bold numbers, in some cases there are strong preferences for threads (N and P strongly prefer threads), but we cannot observe any preferences for

Table 4: Evaluation per each individual MBTI variable. The number of students are shown for each variable and the distribution among messages and threads. Note that for each pair (e.g. E/I), the whole sample set applies and divides into two possibilities (like E and I).

	S 19	N 12	T 26	F 5	J 22	P 9	E 15	I 16	no. in style
Threads	10	9	15	4	12	7	9	10	19
Messages	9	3	11	1	10	2	6	6	12

messages in the converse MBTI variable. Grouping TJ and NOT TJ (Table 6), leads to a clear preference toward threads for NOT TJ (p value < 0.035), but no specific preference in TJ. However, if we compare ST and NT (Table 6), it suggests a slight preference toward messages for ST (Table 7). Both classes, however, have not enough statistical significance (p value is not < 0.05).

Table 5: Raw data of evaluation per MBTI type. In each box, the type, the number of students falling into this category and the number preferring either MPI or Pthreads (no. MPI / no. Pthreads) are shown.

MBTI type	ISTJ	ISFJ	INFJ	INTJ
Test 1: no. in type	3	0	0	1
MPI / Pthreads	2 / 1	-	-	0 / 1
Test 2: no. in type	5	1	0	2
Java sockets / Threads	3 / 2	0 / 1	-	0 / 2
MBTI type	**ISTP**	**ISFP**	**INFP**	**INTP**
Test 1: no. in type	1	0	0	0
MPI / Pthreads	0 / 1	-	-	-
Test 2: no. in type	2	0	0	1
Java sockets / Threads	1 / 1	-	-	0 / 1
MBTI type	**ESTP**	**ESFP**	**ENFP**	**ENTP**
Test 1: no. in type	0	0	1	0
MPI / Pthreads	-	-	0 / 1	-
Test 2: no. in type	1	0	2	1
Java sockets / Threads	0 / 1	-	1 / 1	0 / 1
MBTI type	**ESTJ**	**ESFJ**	**ENFJ**	**ENTJ**
Test 1: no. in type	2	0	0	2
MPI / Pthreads	1 / 1	-	-	1 / 1
Test 2: no. in type	3	1	0	2
Java sockets / Threads	2 / 1	0 / 1	-	1 / 1

Clearer results are obtained if classifying STJ vs. NOT STJ (Table 8), i.e. here we find a slightly clearer bias of STJ toward preferring messages - which, however, is still not statistical significant, though for the converse choice of threads by STJ it is (p value < 0.025). Note that STJ's preference for messages would confirm the expectation of being stronger in separation. But all three S, T and J may have to apply together to establish a preference. Considering the distribution, 42% of students are STJ and 54.9% are ST. Thus, this distribution and the tendency of these groups to prefer message passing, may contribute to the fact that MPI is currently the dominating approach to parallel processing. Since the results in this direction still lack a bit more statistical significance, this suggests future work to obtain a larger sample set. However, we can now conclude with clear significance that NOT STJ and NOT TJ have a very strong bias toward threads, whereas their converse class is at least more balanced if not showing a bias toward messages.

Table 6: Distribution of preferences for threads and messages among TJ and not TJ.

	TJ 20	not TJ 11
Threads	10 (6+4)	9 (7+2)
Messages	10 (6+4)	2 (2+0)

Table 7: Distribution of preferences for threads and messages among ST and NT. Numbers in parenthesis are for Test 2 and Test 1.

	ST 16	NT 9
Threads	7 (5+2)	7 (5+2)
Messages	9 (6+3)	2 (1+1)

Table 8: Distribution of preferences for threads and messages among STJ and not STJ. Numbers in parenthesis are for Test 2 and Test 1.

	STJ 13	not STJ 18
Threads	5 (3+2)	14 (10+4)
Messages	8 (5+3)	4 (3+1)

As regards the answers to Question 2, students mention mostly the synchronization problem as a difficulty in using threads - which conforms to the standard opinion about thread programming.

4 Summary and Conclusion

Shared-data processing via threads is mainly concerned with synchronizing accesses, message passing via MPI or sockets with partitioning and data distribution. Furthermore, thread programming provides more flexibility and variability, whereas message passing via MPI is more restrictive and uniform. Both MPI and Java sockets support a clear separation of data and protected address spaces. Thus, the focus is different as well as the machine types for which they were designed. We have shown that correlations to certain MBTI personality type classes can be established and have given some evidence for the validity of our claims by an empirical study with a number of students implementing programming assignments with different languages. This would also explain why certain groups of people argue for one or the other programming model and prefer to purchase one or the other machine type. In other words, the decision is not made by which approach is "better" (for a certain problem or machine type) but also by what people like better. However, our test was confined to students who first met parallel processing and concurrency. In the future, we intend to increase the sample set to confirm our initial results and enable more detailed classification of MBTI types and problem formulations (especially toward both synchronous and asynchronous execution), to extend our tests toward people experienced in parallel processins, and to include higher-level programming models to explore their gain in productivity for certain personality types.

5 Acknowledgements

Thanks to Karen Y. Fung for explanations of statistical sampling. Furthermore, we thank all students at the University of Windsor who kindly participated in our test.

6 References

[1] J.E. Barbuto and B.A. Plummer. Mental Boundaries and Jung's Psychological Types: A Profile Analysis. Journal of Psychological Type, Vol. 54, 2000, pp. 17-21.

[2] R. Brooks. Studying Programmer Behaviors Experimentally: The Problems of Proper Methodology. Communications of the ACM, 23, 1980, pp. 207-213.

[3] F. Cappello and D. Etiemble. MPI versus MPI+OpenMP on the IBM SP for the NAS Benchmarks. Proc. IEEE/ACM Supercomputing Conf. (SC2000), Dallas/TX, USA, Nov. 2000.

[4] D.E. Campbell and J.M. Kain. Personality Type and Mode of Information Presentation: Preference, Accuracy, and Efficiency in Problem Solving. Journal of Psychological Type, Vol. 20, 1990.

[5] L.F. Capretz. Making Sense of Software Engineering and Personality Types. Submitted.

[6] Y.C. Hu, A. Cox and W. Zwaenepoel. Improving Fine-Grained Irregular Shared-Memory Benchmarks by Data Reordering. Proc. IEEE/ACM Supercomputing Conf. , Dallas/TX, USA, November 2000.

[7] Dongming Jiang and Jaswinder Pal Singh. Scaling Application Performance on a Cache-coherent Multiprocessor. Proc. 26th Int. Symp. on Computer Architecture (ISCA), Atlanta/GE, USA, May, 1999.

[8] C.G. Jung. Psychological Types (H.G. Baynes, translation revised by R.F.C. Hull), Vol. 6 of the collected works of C.G. Jung. University Press, Princeton, NJ/USA,1971 (original work published 1921).

[9] K. Kennedy, C.F. Bender, J.W.D. Connolly, J.L. Hennessy, M.K. Vernon, and L. Smarr. A Nationwide Parallel Computing Environment. Communications of the ACM, Vol. 40, No. 11, 1997, pp. 63--72.

[10] B. Lewis and D.J. Berg. Threads Primer-A Guide to Multithreaded Programming. SunSoft Press, Prentice Hall, Mountain View, CA/USA1996.

[11] M.L. Lyons. The DP Psyche. Datamation, 31(16), August 1985, pp. 103-110.

[12] I.B. Myers and M.H. McCaulley. Manual: A Guide to the development and Use of the Myers-Briggs Type Indicator. Palo Alto, CA/USA, Consulting Psychologists Press, 1985.

[13] MPI web page. http://www-unix.mcs.anl.gov/mpi/, August 2001.

[14] I.B. Myers, M.H. McCaulley, N.L. Quenk, and A.L. Hammer. MBTI Manual: A guide to the development and use of the Myers-Briggs Type Indicator (3rd Edition). Consulting Psychologists Press, Palo Alto, CA/USA, 1998.

[15] D.S. Nikolopoulos, T. S. Papatheodorou, C.D. Polychronopoulos, and E. Ayguarde. Is Data Distribution Necessary in OpenMP. Proc. IEEE/ACM Conf. on Supercomputing Dallas/TX, USA, 2000.

[16] OpenMP web page. http://www.OpenMP.org/, August 2001.

[17] C.M. Pancake. Can Users Play an Effective Role in Parallel Tools Research? Int. Journal of Supercomputing and HPC, Vol. 11, No. 1, 1997, pp. 84-94.

[18] H. Shan, J.P. Singh, L. Oliker and R. Biswas. A Comparison of Three Programming Models for Adaptive Applications for the Origin2000. Proc. IEEE/ ACM Supercomputing Conf., Nov. 2000.

[19] D. Shneiderman. Psychology: Human Factors in Computer and Information Systems. Cambridge, MA, Winthrop Publishers, 1980.

[20] A.C. Sodan. Toward Successful Personal Work and Relations - Applying a Yin/Yang Model for Classification and Synthesis. Social Behavior and Personality -An International Journal. Vol. 27, No. 1, Jan. 1999.

[21] A.C. Sodan. Survey of Runtime-Implementation Strategies for Parallel Systems. Technical Reports 01-01 and 01-11, University of Windsor, Computer Science, Jan. and Nov. 2001. Submitted.

[22] S.P. Springer and G. Deutsch. Left Brain, Right Brain. W.H. Freeman and Company, New York, 1981.

[23] K. Shen, H. Tang and T. Yang. Adaptive Two-level Thread Management for Fast MPI Execution on Shared Memory Machines. Proc. IEEE/ACM Supercomputing Conf. , Seattle/WA, USA, 1999.

[24] B. Wilkinson and M. Allen: Parallel Programming, Prentice Hall, 1999.

Brief Bio

Dr. Angela Sodan has worked many years in a research lab which was focused on parallel processing, has been a Visiting Professor at the University of New Mexico and worked with Sandia Labs and is now an Associate Professor at the University of Windsor. Her research interest is runtime support in parallel processing, especially multithreading and dynamic scheduling in cluster and grid environments.