Bayes Parallel Computation: Choosing the number of processors

Tihomir Asparouhov & Bengt Muthén

April 30, 2019

1 Introduction

This note discusses Bayes computations in Mplus. In version 8.3, a major effort has been made to speed up the Bayesian computations. In some cases, the computing time has been cut in half and even cut to a fifth of the time in Version 8.2. A key part of the success of this work is the use of a new approach of parallelized computing. As background for this development, Section 2 discusses how computational speed is influenced by the number of iteration chains, CPU processors, and analyses run at the same time. Section 3 discusses the new parallelized computing and gives a guide to how it can be most efficiently used on different computers. Section 4 provides timing examples in realistic settings.

2 Chains, Processors, and Copies

Bayesian MCMC estimation is based on multiple chains of iterations computed independently. The number of chains is controlled by the **Chains** option in Mplus. The Mplus default is 2 chains and that choice is optimal in most situations. The choice regarding the number of chains should be driven primarily by these two factors:

1. minimize the number of MCMC iterations needed for convergence
2. minimize the probability of premature or false convergence

Models that are somewhat poorly identified and poorly mixing models can benefit from increasing the number of independent chains in the computation to reduce the probability of false convergence. Generally, however, 2 chains is enough in most cases. Estimation using a single chain usually increases the number

of iterations needed for convergence and it increases the probability of false of premature convergence.

Because the MCMC chains are independent of each other, they can be run in parallel on different processors, that is, on different CPU threads or CPU cores; for a description of technical terms, see

https : *//www.howtogeek.com/*194756*/cpu*−*basics*−*multiple*−*cpus*−*cores*−*and* − *hyper* − *threading* − *explained/*.

This reduces the amount of computational time. In Mplus, the number of processors is controlled by the **Processors** option (referred to as **Proc** here). Note, however, that the Mplus default is 1 processor. Thus by default Mplus will not estimate the chains on different processors unless the **Proc** option is used. **Proc** = 2 has been the recommended setting. Note also that most PCs available on the market today are equipped with multiple processors. For example, most Intel i7 processors have 4 cores and 8 threads. This means that 4 (almost) independent processors are available for use and that number could be as high as 8 in certain computations. The i9-9900K has 8 cores and 16 threads. Therefore, utilizing the **Proc** option, we can easily reduce the computational time. For example, given that a CPU has two cores (processors) or more, and the model estimation is based on the Mplus default of 2 chains, specifying the **Proc=2** option would reduce the computational time by almost a half, simply because the two chains will be executed in parallel, i.e., at the same time rather than sequentially. Note, however, that this assumes that no other work is being performed on the same computer.

Most CPUs have multiple cores (and threads) but these cores are not completely independent of each other. The cores usually share many other components within the computer such as BUS speed, CPU cache, memory, etc. To illustrate this, we conduct the following experiment. Using the User's Guide example 9.32, we record the computational time for 2000 MCMC iterations (per chain) and we vary the number of chains in the estimation and the number of processors in the estimation. We also vary the number of copies of the example run at the same time to illustrate running several analyses at the same time. The experiment is conducted on a Windows machine with CPU i7-4710HQ which features 4 cores and 8 threads. The results are presented in Table 1.

We can see that increasing the number of processors from 1 to 2 (when copy=1 and chains=2) indeed reduced the time substantially from 35 seconds to 19 seconds. This is not quite a 50% reduction for two reasons. Running the two chains in parallel has two sources of overhead. The first one is within Mplus where the chains wait for each other and compare the results every 100 iterations to determine convergence. The second overhead is within the CPU itself because of the fact that the two cores are not completely independent of each other. In this example, both overheads are fairly small and the reduction in computational

Table 1: Computation time in seconds for User's Guide example 9.32

Copies	Chains	Processors	Time
1	2	1	35
1	1	1	18
1	2	2	19
1	3	3	24
1	4	4	30
2	2	2	29
2	1	1	18
4	1	1	27

time due to the parallel computing is 46%, instead of 50% (which would be the case if there was no overhead in the parallel computing).

Next, notice that the time increases dramatically when comparing the cases where Chains=Processors (and copy=1) as the number of chains increases. In this comparison, the time increases from 18 seconds to 30 seconds. This points out that the overhead of parallel computing jumps dramatically beyond the case of 2 chains. This jump in the overhead is mostly due to the CPU and the lack of independence between the cores, i.e., the cores interfere with each other and are unable to effectively utilize the parallel algorithms of Mplus. This particular CPU appears to be able to sustain 2 independent parallel computations with no interference but anything higher than that yields fairly substantial dependence, i.e., the effective number of cores for this CPU is 2.

Fortunately, newer CPUs have less interferences between the cores (i.e. i9 or newer i7 CPUs) due to increases in Cache, BUS speed, memory speed etc. These have a higher number of effective cores (although that number is still lower than what the CPU is advertised with). The level of interference between the cores is CPU specific and it should be tested and analyzed (with a similar experiment) to determine the best possible setup for Mplus analyses.

Table 1 shows the interference between the cores in a different way as well. If we compare the case of (Proc=1, Chain=1) run as one copy, two copies and four copies, we see that running two copies yields the same computational time as running a single copy, but running 4 copies at the same time yields 27 second, i.e., a 9 seconds increase. This is again due to the fact that the CPU can handle 2 processors fairly well but with 4 processors the interference between the processors prevents us from getting a good timing result.

3 New parallel algorithms in Mplus 8.3: Processors > Chains. Nested parallelism.

Prior to Mplus version 8.3, the Bayesian computation in Mplus could be parallelized in only one way. Different chains could be computed by different processors. This means that Mplus was unable to utilize more processors than the number of chains, i.e., using the Mplus default of 2 chains, it was not possible to utilize more than 2 processors to estimate a model. In Mplus 8.3, we have implemented new Bayes algorithms for all models, except cross-classified models, which allow us to use more than one processor for each chain. We refer to this within-chain parallelization as nested parallelism. There are two stages of parallelism nested within each other. First, the algorithm is made parallel as the two MCMC chains are computed in parallel. Second, the computation within each chain is further parallelized so that more than one processor can be utilized for the computations of that chain.

The new parallel within-chain computation works approximately as follows. Suppose that 2 processors are available for computing one MCMC chain. The MCMC computations are by nature very sequential / non-parallel. The algorithm updates one quantity given all other. If, however, two parameters/latent variables/random effects are conditionally independent (given everything else), they can be updated simultaneously on different processors. Thus a new MCMC parallelization algorithm can be implemented where such conditionally independent quantities are identified and distributed across the available processors. Quantities that can not be paired with other quantities under this conditional independence setting are not updated in parallel. Only one processor can be used to update such quantities while the other processors will be waiting for this updating step to occur.

Due to the complexity of the algorithm, the overhead of the nested parallel computation will be substantially larger than the case of the standard parallel computation where Processors=Chains, available in Mplus 8.2 and earlier versions. The second important fact here is that because the stream of the random number generator is different, the final results of the model estimation will be slightly different when nested parallelism is used (note that this is not new given that slightly different results are to be expected with Bayes when different number of iterations are used). Let's illustrate this point with an example. Suppose that a model is estimated with 2 independent chains on two separate processors. Each processor is assigned a random number generation stream, i.e., each chain will be estimated with one random generation stream. When the same model is estimated on 4 processors, each with its own random number generation stream, each chain will be using 2 random number streams and therefore will not produce the same

Table 2: Computation time in seconds for User's Guide example 9.37

Mplus	Processors	CPU1 i7-4710HQ 4 cores 8 threads 2014	CPU2 i7-3770 4 cores 8 threads 2012	CPU3 i7-8700k 6 cores 12 threads 2017	CPU4 i9-9900k 8 cores 16 threads 2018
8.2	2	114	107	72	69
8.3	2	57	58	38	36
8.3	4	54	35	24	22
8.3	6	57	44	25	21
8.3	8	50	36	27	29
8.3	10	64	50	27	33

result as the 2 processor estimation. Of course, if the MCMC chains are run sufficiently long, then the outcomes should be nearly identical.

Note that the improvements of the nested parallelism algorithm come in addition to other Bayes speed improvements in Mplus 8.3 due to faster algorithms not related to parallel computing at all.

We illustrate the new nested parallel algorithm with User's Guide example 9.37, estimated with 2 chains and 3000 MCMC iterations in each chain, where we vary the number of processors and the CPUs. The number of processors should be a multiple of the number of chains. In this illustration we use 2, 4, 6, 8, and 10 processor estimations. We use 4 different CPUs for the illustration and their specifications and release year are given in Table 2. For comparison purposes, we also include the results of Mplus 8.2 with 2 processors. Note that a number of processors higher than 2 does not affect the computational time in Mplus 8.2 and nested parallelism is not available prior to Mplus 8.3.

The timing results presented in Table 2 can be used to draw several conclusions. Using just 2 processors, Mplus 8.3 is almost twice as fast as Mplus 8.2. This timing improvement, however, is model specific. The biggest timing improvements in Mplus 8.3, unrelated to the nested parallelism, are for cross-classified models (see the Figure 1 Example 11 which is UG ex9.39b) and for three-level models (see the Figure 1 Example 8 which is UG mcex9.21). Many other models, however, will be noticeably faster in Mplus 8.3.

Note also that the choice of 10 processors did not result in the fastest computation. CPU1 and CPU2 have 4 cores with 8 threads, i.e., there are not 10 processors on these machines. CPU3 has 6 cores and 12 threads. CPU4 has 8

cores and 16 threads. Essentially CPU3 and CPU4 also do not have 10 processors. This is because threads interfere much more with each other than cores. Using the incorrect number of processors made the computation substantially slower on all the CPUs than with a more optimal choice of the number of processors. The more processors that are used in the computation, the bigger the overhead from the Mplus-MCMC algorithm as well as the overhead from the system parallel computing capability.

These results also show how system dependent the timing results are. Increasing the number of processors from 2 to 4 reduced the computational time by 5% on CPU1, by 40% on CPU2, by 37% on CPU3, and by 39% on CPU4. The optimal number of processors for CPU1 is 8 while for CPU2 and CPU3 it is 4 and for CPU4 it is 6. The older CPU1 and CPU2 displayed no clear monotonic or functional relationship between the number of processors and the computational time, while the newer CPU3 and CPU4 displayed a hyperbolic behavior with the most optimal number of processors being in the middle (the bottom tip of the hyperbola). Thus, the newer processors' behavior is more in line with our understanding of parallel computing. Adding more processors to the computation would generally improve the speed until we add too many processors, at which point the overhead is so great that the timings will start to increase. If X is the number of processors used in the computation, P is the computational time that can be made parallel, S is the computational time that cannot be made parallel, and O is the overhead for every processor, we would expect the total computational time to be the following hyperbolic function

$$\frac{P}{X} + S + X \cdot O. \tag{1}$$

Thus, even if we have an infinite supply of processors, the optimal number of processor which minimizes the above function, i.e. $X = \sqrt{P/O}$, will be a finite number.

Note that the optimal choice for the number of processors also depends on the actual example, i.e., the timing comparison results in Table 2 will be different for a different example. More complex models and models with larger data sets are more likely to benefit from a larger number of processors. Better hardware is much more likely to have more benefit for big computations with large data and large number of variables. Even within the same example, the optimal choice of the number of processors may depend on the output requests and the plot requests specified in the input file. Such requests change the overhead of the parallel computing as steps that are not parallel could be weighted more with output and plot requests.

Note also that the examples we used here for illustration had a fixed number of iterations. If the number of iterations is not set to a fixed number but

is determined by the convergence of the model, then changing the number of processors would change randomly the number of iterations to convergence, which would affect the final computational time.

Because of these complexities it is difficult to universally recommend the number of processors to use. A safe choice appears to be 4 processors with 2 chain computations. This is obtained by simply saying **Proc** = 4 because 2 chains is the default. Older CPUs, however, may not work well even with that. Perhaps the most reliable method for selecting the number of processors to use in the Bayes estimation on a specific computer is to perform a limited experiment as the one we conducted here, using a small number of prefixed iterations.

Another issue that we have noticed is that older CPUs do not consistently produce the same time estimates. Larger timing variations occur with older CPUs than with newer CPUs. This is most likely related to the ability of the CPUs to handle the complexity of the Mplus parallel algorithms. Newer CPUs are clearly better suited to utilize these algorithms than older CPUs. Clearly CPU4 produced the best results for this example, which is expected since this CPU is the latest in the Intel product line, however, CPU3 is not far behind and the difference between the two appears to be marginal. The differences with the newer CPUs and the older CPUs, however, are substantial. We also see from the above results that the effective number of cores for CPU4=i9-9900k is probably closer to 6 (not 8 which is the actual number of cores), while for CPU3=i7-8700k the effective number of cores is probably closer to 4. This means that CPU4 may have bigger advantages over CPU3 for more computationally intensive examples.

4 Timing examples

This section discusses timings for 11 examples where the complexity of the models and the number of iterations correspond to common real-data situations. Table 3 gives the timings for two different computers and using different number of processors specified by the **Proc** option. The i7-7700 computer uses 4 cores while the i9-9900 computer uses 8 cores. For both computers, comparisons can be made to timings of Mplus Version 8.2 which uses **Proc** = 2. The examples are divided into the analysis areas of single-level analysis, two-level analysis, three-level analysis, and cross-classified analysis.

As a separate speed-up effort from the nested parallelism discussed in Section 3, the Bayes timing improvements in version 8.3 are also due to code changes that make several key algorithms more efficient. The effects of more efficient algorithms can be studied by comparing timings of version 8.3 runs using **Proc** = 2 with those of version 8.2 runs using **Proc** = 2. For examples using cross-classified analysis, this is the only source of speed improvements. The effects of

the nested parallelism can be studied by comparing **Proc** > 2 runs with **Proc** $= 2$ runs.

The characteristics of the 11 examples are as follows. The number of iterations was chosen to provide a clear indication of convergence and set so that all runs for a given example use the same number of iterations. The examples are run one at a time with no other activities on the computer.

- Ex1 is a single-level factor analysis with ordinal factor indicators, N=13833, 3000 iterations.
- Ex2 is the UG mcex9.29 two-level factor analysis with covariates and a random factor residual variance modified to 810 clusters, N=8000, 10000 iterations.
- Ex3 is the UG ex 9.37 two-level time series (DSEM) example with 4 random effects including a trend and a random variance, N=200, T=100, 10000 iterations.
- Ex4 is a two-level time series (RDSEM) example with 9 random effects including 2 random variances, N=230, T=100, 2000 iterations.
- Ex5 is a two-part time series (RDSEM) example with trend and a binary level-2 outcome, N=230, T=100, 2500 iterations.
- Ex6 is a two-part time series (RDSEM), N=230, T=100, 2000 iterations.
- Ex7 is the UG ex mcex9.21 three-level mediation model with 10 replications, 50 level 3 clusters, 30 level 2 clusters, N=7500, 1000 iterations.
- Ex8 is the UG ex mcex9.26 cross-classified IRT model with 50 random items, 10000 subjects, 1000 iterations.
- Ex9 is the UG ex 9.27 cross-classified factor model with a trend, random loadings and intercepts and 75 subjects, T=100, 3000 iterations.
- Ex10 is the UG ex9.38b cross-classified regression model with random slope, random residual variance, and random AR(1), N=200, T=100, 2000 iterations.
- Ex11 is the UG ex9.39b cross-classified regression model with random slope, random residual variance, random AR(1), and a linear trend, N=200, T=100, 7800 iterations.

The top section of Table 3 shows results for the older and slower i7 computer and the bottom section shows results for the newer and faster i9 computer. In each section, the first column shows the timings for version 8.2 which can benefit from only **Proc**=2. The other columns show results for version 8.3. Also shown for each computer is the percentage time used by version 8.3 with different number of processors relative to the time used by version 8.2 with **Proc** = 2. The results will first be discussed for **Proc** values not exceeding the number of cores for the two computers, that is, 2 and 4 for the i7 computer and 2, 4, 6, and 8 for the i9 computer.

Example 1 shows that for the i7, the **Proc**=4 run takes only 61% of the time it takes in version 8.2. The i9 **Proc**=4, 6, and 8 runs show the increasing benefits from the nested parallelization, reducing the time to 60, 55, and 50%, respectively of the version 8.2 **Proc**=2 run.

The Example 2 runs for the i9 show that **Proc**=6 and 8 can not be universally recommended in that for this example, **Proc**=4 gives the largest reduction in time. As discussed in Section 3, in an example like this where each iteration is rather fast, the overhead for the nested parallelization will be relatively bigger. The same feature is seen for **Proc**=8 in the relatively fast Example 3. In contrast, the slower examples in this two-level time series section have timing improvements for both **Proc**=6 and 8, even cutting the time to 20% for Example 5 with **Proc**=8.

For the cross-classified examples 8 - 11, no timings are given for **Proc** $>$ 2 because no nested parallelization has been made. The effects of using more efficient algorithms are seen in the **Proc**=2 columns. For example, Example 11 shows a reduction in time to 25% for both the i7 and i9 computer.

When **Proc** exceeds the number of cores of the two computers, the results are more unpredictable. There are examples where the use of more processors results in better timings. With the i7 computer, Example 1 is faster for **Proc**=6 and 8 than for 4. With **Proc**=8, the time relative to **Proc**=2 is 46%. For Example 6 with **Proc**=8, 44% is obtained. For Example 7 with **Proc**=8, 36% is obtained. Example 5, however, sees an increase in time when using **Proc**=6, in this case mainly due to using considerably more iterations. With the i9 computer, several examples get worse timings with **Proc** greater than the number of cores, that is, 10 and 12. An exception is Example 1 which is faster using **Proc**= 10 and 12.

In summary, most of the benefit from i9 over i7 is due to being able to use more processors more efficiently. For 2 and 4 processors, the percentage improvement in the i9 over the i7 is not that large. For both the i7 and the i9, **Proc**=4 gives a substantial time improvement over **Proc**=2. Several, more time-consuming examples, benefit substantially from being able to effectively use **Proc**=6 and 8 with the i9. For less time-consuming examples, **Proc**=4 is a safer choice.

	i7-7700k				
	8.2 Proc=2	**8.3 Proc=2**	**8.3 Proc=4**	**8.3 Proc=6**	**8.3 Proc=8**
			Single-level:		
Ex1 (old 8)	00:12:32	00:12:12 (97%)	00:07:38 (61%)	00:06:52 (55%)	00:05:43 (46%)
			Two-level:		
Ex2 (mcex9.29)	00:02:46	00:02:01 (73%)	00:01:46 (64%)	00:01:48 (65%)	00:01:48 (65%)
			Two-level, time series (DSEM, RDSEM):		
Ex3 (UG ex9.37)	00:01:22	00:00:46 (56%)	00:00:40 (49%)	00:00:51 (62%)	00:00:40 (49%)
Ex4 (old 5)	01:42:38	01:08:02 (66%)	00:41:57 (41%)	00:46:10 (45%)	00:42:05 (41%)
Ex5 (old 6)	03:30:56	02:18:52 (66%)	01:31:05 (43%)	01:58:21 (56%)	01:04:15 (30%)
Ex6 (old 7)	00:52:59	00:39:36 (75%)	00:28:48 (54%)	00:26:16 (50%)	00:23:05 (44%)
			Three-level:		
Ex7 (mcex9.21)	00:01:55	00:00:58 (50%)	00:00:55 (48%)	00:00:49 (43%)	00:00:41 (36%)
			Cross-classified:		
Ex8 (mcex9.26)	00:11:06	00: 07:32 (68%)	-	-	-
			Cross-classified, time series:		
Ex9 (UG ex9.27)	00:01:13	00:06:20 (62%)	-	-	-
Ex10 (UG ex9.38b)	01:10:32	00:24:27 (35%)	-	-	-
Ex11 (UG ex9.39b)	08:33:52	02:07:10 (25%)	-	-	-

	i9-9900k						
	8.2 Proc=2	**8.3 Proc=2**	**8.3 Proc=4**	**8.3 Proc=6**	**8.3 Proc=8**	**8.3 Proc=10**	**8.3 Proc=12**
				Single-level:			
Ex1 (old 8)	00:12:11	00:11:28 (94%)	00:07:21 (60%)	00:06:40 (55%)	00:06:02 (50%)	00:05:41 (47%)	00:05:19 (44%)
				Two-level:			
Ex2 (mcex9.29)	00:02:33	00:01:53 (74%)	00:01:34 (61%)	00:02:12 (86%)	00:02:35 (101%)	00:02:41 (105%)	00:02:45 (108%)
				Two-level, time series (DSEM, RDSEM):			
Ex3 (UG ex9.37)	00:01:16	00:00:42 (55%)	00:00:33 (43%)	00:00:30 (39%)	00:00:44 (58%)	00:00:45 (59%)	00:00:47 (62%)
Ex4 (old 5)	01:36:35	01:01:51 (64%)	00:38:57 (40%)	00:28:46 (30%)	00:24:19 (25%)	00:29:11 (30%)	00:28:32 (30%)
Ex5 (old 6)	03:13:19	02:08:10 (66%)	01:25:04 (44%)	01:14:53 (39%)	00:37:55 (20%)	00:44:23 (23%)	00:42:48 (22%)
Ex6 (old 7)	00:49:02	00:37:19 (76%)	00:23:04 (47%)	00:16:42 (34%)	00:14:15 (29%)	00:16:50 (34%)	00:15:54 (32%)
				Three-level:			
Ex7 (mcex9.21)	00:01:47	00:00:54 (50%)	00:00:41 (38%)	00:00:38 (36%)	00:00:47 (44%)	00:00:54 (50%)	00:00:44 (41%)
				Cross-classified:			
Ex8 (mcex9.26)	00:10:01	00:06:26 (64%)	-	-	-	-	-
				Cross-classified, time series:			
Ex9 (UG ex9.27)	00:07:48	00:04:49 (62%)	-	-	-	-	-
Ex10 (UG ex9.38b)	01:06:21	00:23:26 (35%)	-	-	-	-	-
Ex11 (UG ex9.39b)	07:52:43	02:00:05 (25%)	-	-	-	-	-

Table 3: Timing comparisons for two computers, version 8.2, version 8.3, and different number of processors