PROYECTO HPC

NAS PARALLEL BENCHMARKS

Marc Vernet
Paral·lelisme i Sistemes Distribuïts
UPC
04/2020

Evaluación inicial

En las siguientes tablas se ha recogido el tiempo de ejecución de 3 aplicaciones de NAS Parallel Benchmark: BT-MZ, SP-MZ Y LU-MZ. Para cada aplicación se han probado diferentes configuraciones con 4 y 8 nodos. En el caso mpi sólo hay 1 thread por tarea, por lo que no se comparten recursos y hace falta comunicadores mpi . En el caso openmp hay una tarea por nodo, por lo que se comparten recurso. Los casos intermedios son casos mixtos a medio camino entre mpi completo y openmp completo.

En cada caso se ha repetido la ejecución 3 veces seguidas en un mismo nodo para poder calcular una media. En el caso de LU-MZ algunos casos no se han podido calcular, ya el programa daba error con algunas configuraciones concretas.

Los tiempos en ejecución secuencial utilizados para calcular el speedup son:

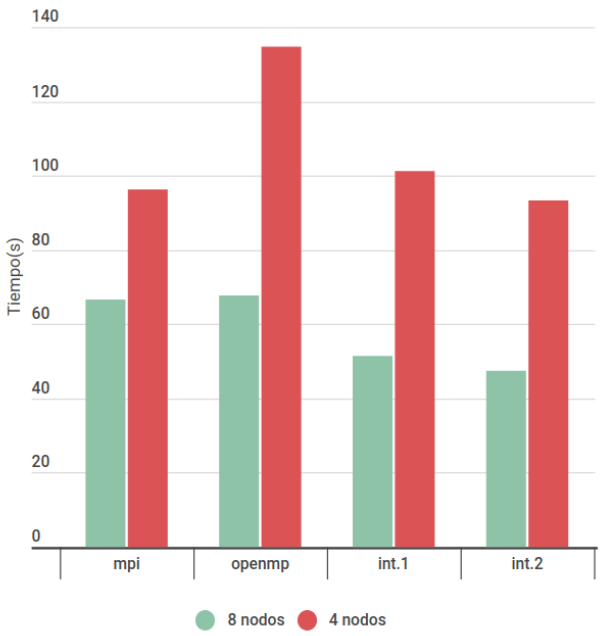
Código	**T(1) (s)**
BT-MZ	18191
SP-MZ	12665
LU-MZ	29431

BT-MZ	nodos	tasks	thread/task	obs. 1	obs.2 (s)	obs.3 (s)	media (s)	speedup
mpi	4	192	1	95.47	97.94	95.43	**96.28**	**188.93**
	8	384	1	66.06	65.95	67.83	**66.61**	**273.09**
openmp	4	4	48	133.3	133.3	137.69	**134.76**	**134.98**
	8	8	48	68.5	69	65.72	**67.74**	**268.54**
int. 1	4	8	24	102.20	100.82	100.79	**101.27**	**179.62**
	8	16	24	51.55	51.45	51.33	**51.44**	**353.63**
int. 2	4	16	12	93.27	93.41	93.24	**93.30**	**194.97**
	8	32	12	47.45	47.49	47.48	**47.47**	**383.21**

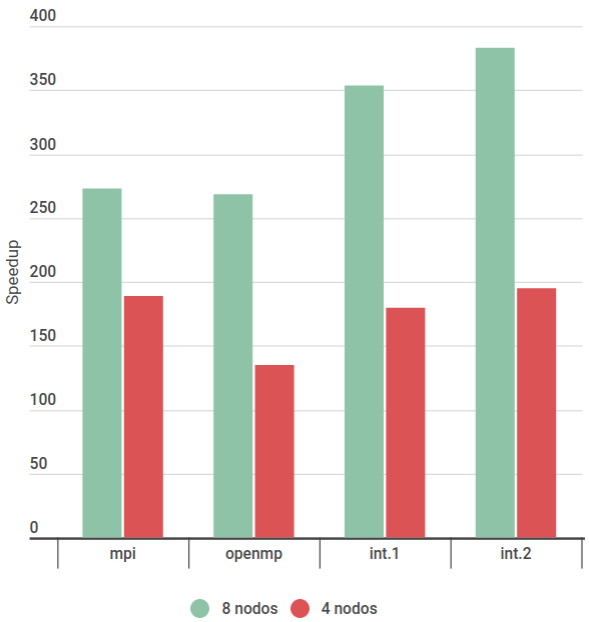
SP-MZ	nodos	tasks	thread/task	obs. 1 (s)	obs. 2 (s)	obs.3 (s)	media (s)	speedup
mpi	4	192	1	71.86	73.52	71.82	**72.4**	**174.93**
	8	384	1	37.05	37.16	36.92	**37.04**	**341.92**
openmp	4	4	48	125.79	127.57	125.36	**126.24**	**100.32**
	8	8	48	66.66	65.59	64.05	**65.43**	**193.56**
int. 1	4	8	24	63.78	63.85	63.77	**63.8**	**198.51**
	8	16	24	32.86	32.29	32.59	**32.58**	**388.73**
int. 2	4	16	12	53.43	52.63	52.58	**52.88**	**239.50**
	8	32	12	27.70	27.75	27.67	**27.7**	**457.22**

LU-MZ	nodos	tasks	thread/task	obs. 1 (s)	obs. 2 (s)	obs.3 (s)	media (s)	speedup
mpi	4	192	1	-	-	-	-	-
	8	384	1	-	-	-	-	-
openmp	4	4	48	164.23	161.18	162.06	**162.49**	**181.12**
	8	8	48	80.81	81.84	80.86	**81.17**	**362.58**
int. 1	4	8	24	147.56	145.67	144.94	**146.05**	**201.51**
	8	16	24	73.11	73.20	73.19	**73.16**	**402.28**
int. 2	4	16	12	150.65	150.59	151.04	**150.76**	**195.21**
	8	32	12	68.73	69.04	69.79	**69.18**	**425.42**

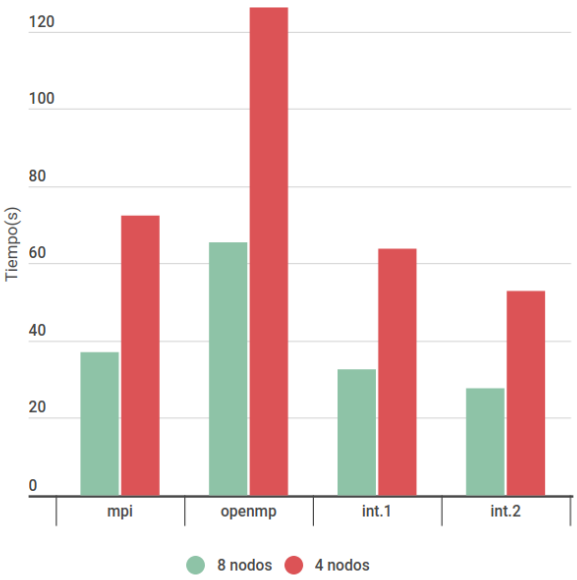
BT-MZ Tiempo de ejecución



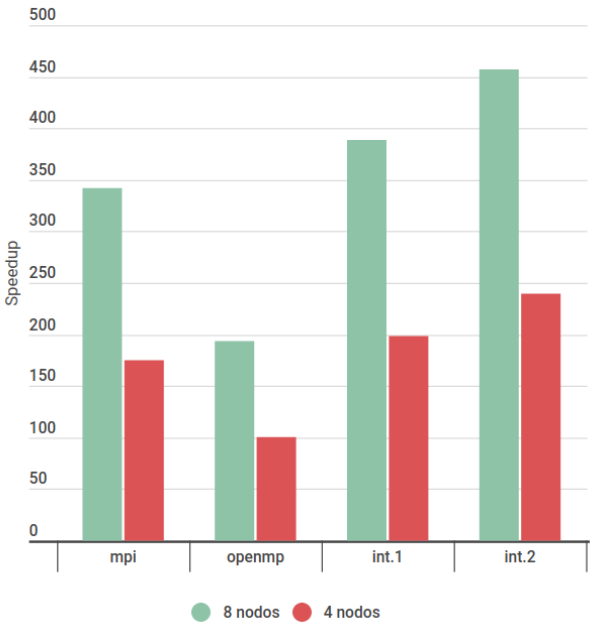
BT-MZ Speedup

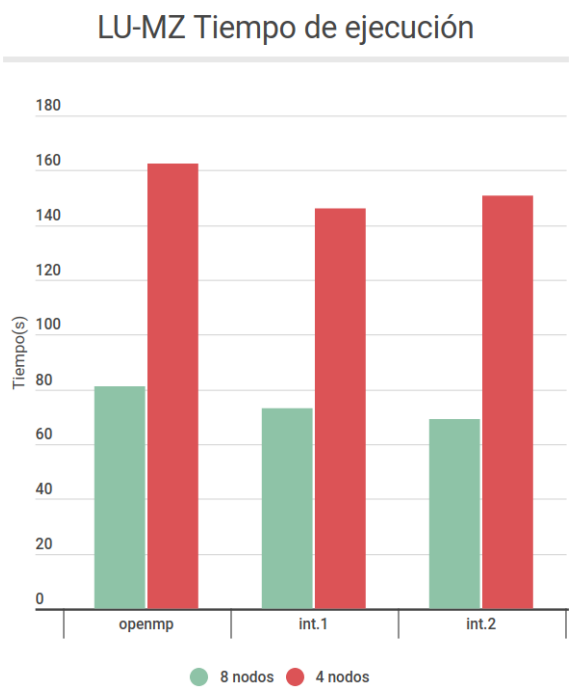


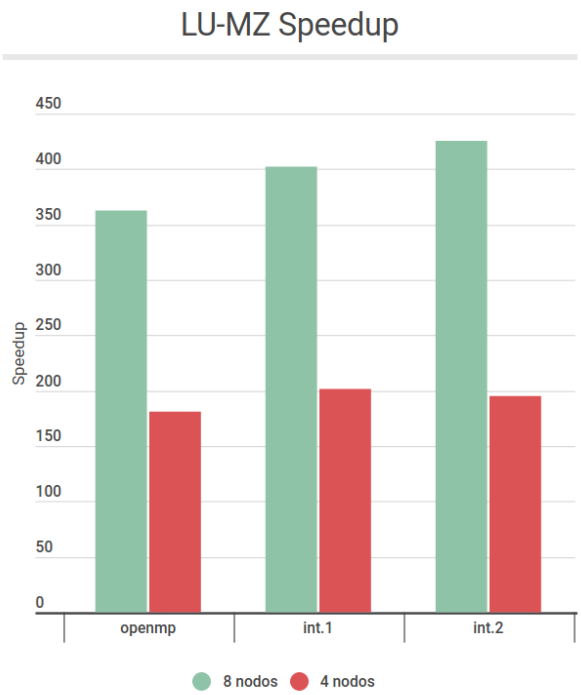
SP-MZ Tiempo de ejecución



SP-MZ Speedup







Obtención de más información mediante slurm

En este apartado se obtiene más información sobre la ejecución de las aplicaciones. Se ha utilizado el comando:

sacct--format=JobID,CPUTime,NCPUS,ReqCPUS,ConsumedEnergy,AveCPUFreq,JobName,Elapsed -p

Con --format podemos elegir qué información queremos recibir en el output, y -p (parseable) hace que el output sea más fácil de leer.

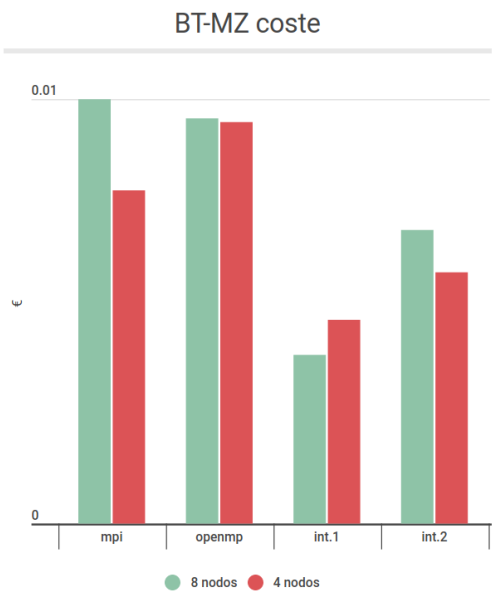
BT-MZ		JobID|CPUTime|NCPUS|ReqCPUS|ConsumedEnergy|AveCPUFreq|JobName|Elapsed	Watts
mpi	4 nodos 192 tasks 1 thread/task	9535136.0|05:13:36|192|192|138.69K|2.08G|bt-mz.D.x|00:01:38| 9535136.1|05:16:48|192|192|141.53K|2.03G|bt-mz.D.x|00:01:39| 9535136.2|05:20:00|192|192|144.15K|2.01G|bt-mz.D.x|00:01:40|	**2318,96**
	8 nodos 384 tasks 1 thread/task	9535242.0|07:15:12|384|384|177.22K|2.06G|bt-mz.D.x|00:01:08| 9535242.1|07:28:00|384|384|182.69K|1.90G|bt-mz.D.x|00:01:10| 9535242.2|07:21:36|384|384|183.23K|2.09G|bt-mz.D.x|00:01:09|	**2967,97**
openmp	4 nodos 4 tasks 48 threads/task	9535311.0|07:12:00|192|192|166.55K|12.34M|bt-mz.D.x|00:02:15| 9535311.1|07:08:48|192|192|168.78K|12.43M|bt-mz.D.x|00:02:14| 9535311.2|07:28:00|192|192|175.78K|12.36M|bt-mz.D.x|00:02:20|	**2792,95**
	8 nodos 8 tasks 48 threads/task	9535328.0|07:28:00|384|384|170.61K|12.23M|bt-mz.D.x|00:01:10| 9535328.1|07:28:00|384|384|173.76K|12.37M|bt-mz.D.x|00:01:10| 9535328.2|07:15:12|384|384|171.70K|12.30M|bt-mz.D.x|00:01:08|	**2820,05**
int. 1	4 nodos 8 tasks 24 threads/task	9536086.0|05:29:36|192|192|127.89K|9.16M|bt-mz.D.x|00:01:43| 9536086.1|05:26:24|192|192|32.62K|9.26M|bt-mz.D.x|00:01:42| 9536086.2|05:36:00|192|192|99.12K|9.74M|bt-mz.D.x|00:01:45|	**1418,85**
	8 nodos 16 tasks 24 threads/task	9536338.0|05:39:12|384|384|81.43K|87.49M|bt-mz.D.x|00:00:53| 9536338.1|05:39:12|384|384|66.55K|87.47M|bt-mz.D.x|00:00:53| 9536338.2|05:39:12|384|384|66.54K|87.48M|bt-mz.D.x|00:00:53|	**1172,29**
int. 2	4 nodos 16 tasks 12 threads/task	9550308.0|05:07:12|192|192|96.00K|174.59M|bt-mz.D.x|00:01:36| 9550308.1|05:04:00|192|192|96.54K|174.69M|bt-mz.D.x|00:01:35| 9550308.2|05:04:00|192|192|127.57K|174.59M|bt-mz.D.x|00:01:35|	**1749,23**
	8 nodos 32 tasks 12 threads/task	9550538.0|05:13:36|384|384|130.26K|174.91M|bt-mz.D.x|00:00:49| 9550538.1|05:13:36|384|384|114.56K|174.97M|bt-mz.D.x|00:00:49| 9550538.2|05:13:36|384|384|130.39K|174.78M|bt-mz.D.x|00:00:49|	**2043,71**

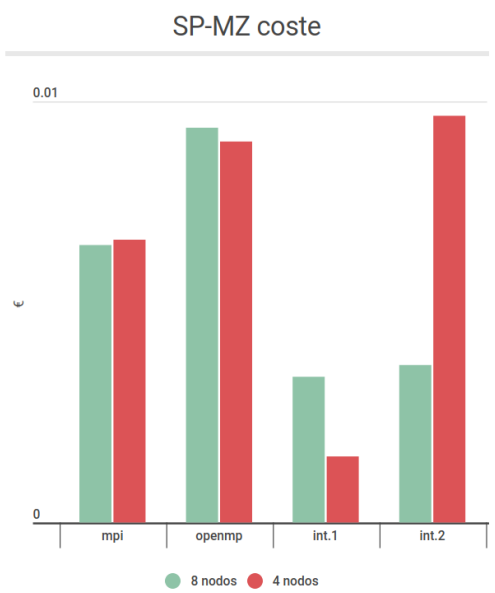
SP-MZ		JobID|CPUTime|NCPUS|ReqCPUS|ConsumedEnergy|AveCPUFreq|JobName|Elapsed	Watts
mpi	4 nodos 192 tasks 1 thread/task	9535144.0|03:56:48|192|192|121.13K|2.08G|sp-mz.D.x|00:01:14| 9535144.1|03:56:48|192|192|121.25K|2.08G|sp-mz.D.x|00:01:14| 9535144.2|03:53:36|192|192|121.08K|2.05G|sp-mz.D.x|00:01:13|	**1983,60**
	8 nodos 384 tasks 1 thread/task	9535251.0|04:09:36|384|384|123.95K|2.10G|sp-mz.D.x|00:00:39| 9535251.1|04:09:36|384|384|108.49K|2.10G|sp-mz.D.x|00:00:39| 9535251.2|04:03:12|384|384|125.06K|2.10G|sp-mz.D.x|00:00:38|	**1945,35**
openmp	4 nodos 4 tasks 48 threads/task	9535316.0|06:52:48|192|192|162.67K|12.34M|sp-mz.D.x|00:02:09| 9535316.1|06:49:36|192|192|163.99K|12.31M|sp-mz.D.x|00:02:08| 9535316.2|06:49:36|192|192|164.82K|12.35M|sp-mz.D.x|00:02:08|	**2672,13**
	8 nodos 8 tasks 48 threads/task	9535331.0|07:08:48|384|384|169.01K|11.95M|sp-mz.D.x|00:01:07| 9535331.1|07:02:24|384|384|169.86K|12.32M|sp-mz.D.x|00:01:06| 9535331.2|07:08:48|384|384|169.65K|12.28M|sp-mz.D.x|00:01:07|	**2770,49**
int1	4 nodos 8 tasks 24 threads/task	9536096.0|03:28:00|192|192|21.56K|87.43M|sp-mz.D.x|00:01:05| 9536096.1|03:28:00|192|192|43.00K|87.41M|sp-mz.D.x|00:01:05| 9536096.2|03:31:12|192|192|21.33K|87.41M|sp-mz.D.x|00:01:06|	**464,48**
	8 nodos 16 tasks 24 threads/task	9536357.0|03:44:00|384|384|66.70K|87.47M|sp-mz.D.x|00:00:35| 9536357.1|03:31:12|384|384|66.63K|87.46M|sp-mz.D.x|00:00:33| 9536357.2|03:50:24|384|384|55.62K|87.45M|sp-mz.D.x|00:00:36|	**1021,85**
int2	4 nodos 16 tasks 12 threads/task	9550310.0|02:52:48|192|192|77.32K|174.86M|sp-mz.D.x|00:00:54| 9550310.1|02:52:48|192|192|58.98K|174.68M|sp-mz.D.x|00:00:54| 9550310.2|02:56:00|192|192|78.85K|175M|sp-mz.D.x|00:00:55|	**2852,45**
	8 nodos 32 tasks 12 threads/task	9550550.0|03:05:36|384|384|80.74K|175M|sp-mz.D.x|00:00:29| 9550550.1|03:05:36|384|384|61.48K|175M|sp-mz.D.x|00:00:29| 9550550.2|03:05:36|384|384|61.36K|175M|sp-mz.D.x|00:00:29|	**1103,82**

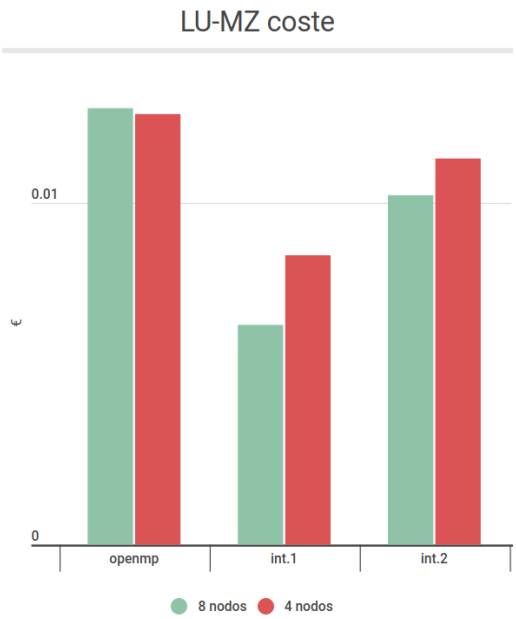
LU-MZ		JobID|CPUTime|NCPUS|ReqCPUS|ConsumedEnergy|AveCPUFreq|JobName|Elapsed	Watts
openmp	4 nodos 4 tasks 48 threads/task	9535322.0|09:07:12|192|192|233.68K|6.10M|lu-mz.D.x|00:02:51| 9535322.1|08:48:00|192|192|229.09K|6.32M|lu-mz.D.x|00:02:45| 9535322.2|08:51:12|192|192|230.49K|6.18M|lu-mz.D.x|00:02:46|	**3726,77**
	8 nodos 8 tasks 48 threads/task	9535334.0|08:51:12|384|384|228.86K|18.40M|lu-mz.D.x|00:01:23| 9535334.1|08:51:12|384|384|230.57K|12.47M|lu-mz.D.x|00:01:23| 9535334.2|08:57:36|384|384|233.50K|12.34M|lu-mz.D.x|00:01:24|	**3775,95**
int1	4 nodos 8 tasks 24 threads/task	9535250.0|07:47:12|192|192|169.57K|30.73M|lu-mz.D.x|00:02:26| 9535250.1|07:50:24|192|192|114.73K|30.86M|lu-mz.D.x|00:02:27| 9535250.2|08:00:00|192|192|175.88K|29.68M|lu-mz.D.x|00:02:30|	**2502,73**
	8 nodos 16 tasks 24 threads/task	9536349.0|08:00:00|384|384|115.60K|87.43M|lu-mz.D.x|00:01:15| 9536349.1|08:00:00|384|384|175.35K|87.44M|lu-mz.D.x|00:01:15| 9536349.2|07:53:36|384|384|58.70K|87.43M|lu-mz.D.x|00:01:14|	**1901,63**
int2	4 nodos 16 tasks 12 threads/task	9550311.0|08:09:36|192|192|182.35K|174.67M|lu-mz.D.x|00:02:33| 9550311.1|08:09:36|192|192|183.99K|174.67M|lu-mz.D.x|00:02:33| 9550311.2|08:09:36|192|192|246.44K|174.65M|lu-mz.D.x|00:02:33|	**3338,79**
	8 nodos 32 tasks 12 threads/task	9550551.0|07:47:12|384|384|205.64K|174.84M|lu-mz.D.x|00:01:13| 9550551.1|07:34:24|384|384|175.28K|174.78M|lu-mz.D.x|00:01:11| 9550551.2|07:28:00|384|384|173.06K|174.84M|lu-mz.D.x|00:01:10|	**3021,85**

	BT-MZ		SP-MZ		LU-MZ	
	Watts	€	Watts	€	Watts	€
mpi 4 nodos	2318,96	**0.00785**	1983,60	**0.00672**	-	-
mpi 8 nodos	2967,97	**0.01**	1945,35	**0.00659**	-	-
openmp 4 nodos	2792,95	**0.00946**	2672,13	**0.00905**	3726,77	**0.01262**
openmp 8 nodos	2820,05	**0.00955**	2770,49	**0.00938**	3775,95	**0.01279**
int.1 4 nodos	1418,85	**0.00480**	464,48	**0.00157**	2502,73	**0.00848**
int. 1 8 nodos	1172,29	**0.003972**	1021,85	**0.00346**	1901,63	**0.00644**
int. 2 4 nodos	1749,23	**0.00592**	2852,45	**0.00966**	3338,79	**0.011314**
int.2 8 nodos	2043,71	**0.00692**	1103,82	**0.00374**	3021,85	**0.01024**

Podemos calcular los Watts de energía consumida a partir de la energía media en Joules que nos proporciona el comando slurm sacct. Si tomamos como referencia un coste de 0.2€Watts/h podemos calcular el coste aproximado de cada ejecución.







Análisis interno de aplicaciones

SP-MZ

Si se observa el porcentaje de tiempo dedicado a comunicaciones (%comm), podemos ver si el código está bien balanceado. En los dos casos mpi, el desbalanceo es muy evidente, ya que hay una clara diferencia entre la región del código con más comunicaciones (%comm max) y la con menos (%comm min). En los casos openmp y los casos intermedios, la diferencia es mucho menor, por lo que se pueden considerar bien balanceados.

En todos los casos, la mayoría de comunicaciones son de sincronización: # MPI_Waitall, #MPI_Barrier. Abundan las comunicaciones colectivas: # MPI_Allgather, # MPI_Bcast. También hay algunas comunicaciones asíncronas: # MPI_Isend, # MPI_Irecv.

Las diferencias que aportan las diferentes configuraciones del código se pueden ver en %wall(). En los casos mpi es mucho más alto, ya que al utilizar un thread por tarea no hay tantos recursos compartidos y hace falta más comunicación. En los casos openmp y intermedios, ya que se utilizan más de un thread por tarea los recursos están más compartidos y las comunicaciones necesarias son menores.

Si se hace un análisis por regiones podemos ver que las funciones principales (compute_rhs, txinvr, x_solve, y_solve, z_solve) son completamente paralelizables, ya que no requieren comunicaciones. Todas las comunicaciones del programa se encuentran fuera de las funciones principales.

BT-MZ

Observando las diferencias de tiempo dedicado a comunicaciones podemos ver si el código está bien balanceado. En la mayoría de configuraciones se puede considerar que hay un buen balanceo, con la excepción del caso mpi de 8 nodos. Las configuraciones con mejor balanceo son los casos intermedios.

En todos los casos, la mayoría de comunicaciones son de sincronización (#MPI_Waitall, #MPI_Barrier). En este código tienen más importancia las comunicaciones punto a punto (#MPI_Isend, #MPI_Irecv), aunque también podemos encontrar comunicaciones colectivas (#MPI_Allgather, # MPI_Bcast). También hay algunas comunicaciones asíncronas (#MPI_Isend, #MPI_Irecv).

Observando el %wall se puede ver cómo afectan las diferentes configuraciones del código a las comunicaciones necesarias. En este programa, el cambio en %wall en función de las características del caso es muy pequeño. Aun así, se puede observar que el caso mpi de 8 nodos tiene un %wall muy elevado, posiblemente consecuencia del desbalanceo que tiene (diferencia grande entre %comm min y %comm max).

Si se hace un análisis por regiones podemos ver que las funciones principales (compute_rhs, x_solve, y_solve, z_solve) son completamente paralelizables, ya que no requieren comunicaciones. Todas las comunicaciones del programa se encuentran fuera de las funciones principales.

LU-MZ

Todas las configuraciones pueden considerarse bien balanceadas, las diferencias entre el mínimo y máximo de comunicaciones son muy pequeñas. El caso con más desbalanceo es el openmp de 8 nodos.

La mayoría de comunicaciones son de sincronización (#MPI_Waitall, #MPI_Barrier). En este código tienen mucha más importancia las comunicaciones punto a punto (#MPI_Isend, #MPI_Irecv), aunque también podemos encontrar algunas comunicaciones colectivas (#MPI_Allgather, # MPI_Bcast). También destacan las comunicaciones asíncronas (#MPI_Isend, #MPI_Irecv).

Observado %wall podemos ver que en este código las configuraciones tienen muy poco efecto en el tiempo destinado a comunicaciones. El caso con un %wall más elevado es el caso openmp de 8 nodos, seguramente debido a que es también el caso con más desbalanceo.

Si se hace un análisis por regiones podemos ver que la mayoría de comunicaciones necesarias son en la función exch_qbc (timestep), que es el bucle principal del programa.

BT-MZ		%comm min	%comm max	%wall	funciones mpi (%mpi)	
mpi	4 nodos 192 tasks 1 thread/task	2.84383	9.38216	5.93	# MPI_Waitall # MPI_Init # MPI_Barrier # MPI_Irecv # MPI_Isend # MPI_Bcast # MPI_Allgather # MPI_Comm_split # MPI_Scan # MPI_Wtime # MPI_Finalize # MPI_Comm_size # MPI_Comm_rank	73.88 23.75 0.80 0.43 0.40 0.30 0.24 0.17 0.03 0.00 0.00 0.00 0.00
	8 nodos 384 tasks 1 thread/task	4.16043	37.8941	33.8	# MPI_Waitall # MPI_Init # MPI_Barrier # MPI_Comm_split # MPI_Isend # MPI_Irecv # MPI_Allgather # MPI_Bcast # MPI_Scan # MPI_Finalize # MPI_Wtime # MPI_Comm_size # MPI_Comm_rank	86.88 11.82 0.99 0.06 0.06 0.06 0.05 0.04 0.04 0.00 0.00 0.00 0.00
openmp	4 nodos 4 tasks 48 thread/task	39.5322	50.9558	8.02	# MPI_Waitall # MPI_Init # MPI_Barrier # MPI_Isend # MPI_Scan # MPI_Irecv # MPI_Bcast	96.83 2.37 0.29 0.22 0.15 0.11 0.02

					# MPI_Comm_split # MPI_Allgather # MPI_Wtime # MPI_Finalize # MPI_Comm_size # MPI_Comm_rank	0.01 0.00 0.00 0.00 0.00 0.00
	8 nodos 8 tasks 48 thread/task	6.41591	10.4321	8.34	# MPI_Waitall # MPI_Init # MPI_Barrier # MPI_Irecv # MPI_Isend # MPI_Scan # MPI_Bcast # MPI_Comm_split # MPI_Allgather # MPI_Wtime # MPI_Finalize # MPI_Comm_rank # MPI_Comm_size	93.33 5.38 0.62 0.23 0.22 0.11 0.07 0.04 0.00 0.00 0.00 0.00 0.00
int.1	4 nodos 8 tasks 24 thread/task	1.47845	4.41202	2.71	# MPI_Waitall # MPI_Init # MPI_Barrier # MPI_Irecv # MPI_Isend # MPI_Scan # MPI_Bcast # MPI_Comm_split # MPI_Allgather # MPI_Finalize # MPI_Wtime # MPI_Comm_rank # MPI_Comm_size	86.38 12.23 0.43 0.39 0.24 0.22 0.07 0.04 0.00 0.00 0.00 0.00 0.00
	8 nodos 16 tasks 24 threads/task	3.22507	4.71851	3.77	# MPI_Waitall # MPI_Init # MPI_Irecv # MPI_Barrier # MPI_Isend # MPI_Scan # MPI_Bcast # MPI_Comm_split # MPI_Allgather # MPI_Finalize # MPI_Wtime # MPI_Comm_rank # MPI_Comm_size	79.98 18.29 0.54 0.52 0.33 0.18 0.13 0.02 0.01 0.00 0.00 0.00 0.00

SP-MZ		%comm min	%comm max	%wall	funciones mpi (%mpi)	
mpi	4 nodos 192 tasks 1 thread/task	3.25505	29.4918	17.95	# MPI_Waitall # MPI_Init # MPI_Barrier # MPI_Irecv # MPI_Comm_split # MPI_Allgather # MPI_Isend # MPI_Bcast # MPI_Scan # MPI_Wtime # MPI_Finalize # MPI_Comm_size # MPI_Comm_rank	89.13 9.68 0.68 0.24 0.10 0.07 0.04 0.04 0.02 0.00 0.00 0.00 0.00
	8 nodos 384 tasks 1 thread/task	10.9951	54.4045	25.01	# MPI_Waitall # MPI_Init # MPI_Barrier # MPI_Irecv # MPI_Comm_split # MPI_Allgather # MPI_Bcast # MPI_Isend # MPI_Scan # MPI_Wtime # MPI_Finalize # MPI_Comm_size # MPI_Comm_rank	86.90 0.64 0.19 0.13 0.12 0.11 0.08 0.03 0.00 0.00 0.00 0.00 0.00
openmp	4 nodos 4 tasks 48 thread/task	4.02309	9.61054	6.98	# MPI_Waitall # MPI_Init # MPI_Isend # MPI_Barrier # MPI_Scan # MPI_Irecv # MPI_Bcast # MPI_Comm_split # MPI_Allgather # MPI_Finalize # MPI_Wtime # MPI_Comm_size # MPI_Comm_rank	96.46 2.82 0.27 0.19 0.11 0.10 0.05 0.01 0.00 0.00 0.00 0.00 0.00

	8 nodos 8 tasks 48 thread/task	6.26537	11.1637	8.56	# MPI_Waitall # MPI_Init # MPI_Barrier # MPI_Isend # MPI_Irecv # MPI_Bcast # MPI_Scan # MPI_Comm_split # MPI_Allgather # MPI_Wtime # MPI_Finalize # MPI_Comm_size # MPI_Comm_rank	94.19 4.90 0.33 0.23 0.20 0.09 0.05 0.01 0.00 0.00 0.00 0.00 0.00
int.1	4 nodos 8 tasks 24 thread/task	2.06477	3.82619	3.23	# MPI_Waitall # MPI_Init # MPI_Barrier # MPI_Isend # MPI_Irecv # MPI_Scan # MPI_Bcast # MPI_Comm_split # MPI_Allgather # MPI_Finalize # MPI_Wtime # MPI_Comm_size # MPI_Comm_rank	83.16 14.60 0.47 0.36 0.22 0.12 0.03 0.01 0.00 0.00 0.00 0.00 0.00
	8 nodos 16 tasks 24 thread/task	4.54543	7.47989	6.09	# MPI_Waitall # MPI_Init # MPI_Barrier # MPI_Isend # MPI_Irecv # MPI_Bcast # MPI_Comm_split # MPI_Scan # MPI_Allgather # MPI_Finalize # MPI_Wtime # MPI_Comm_size # MPI_Comm_rank	81.09 16.65 1.11 0.36 0.36 0.16 0.14 0.13 0.01 0.00 0.00 0.00 0.00

LU-MZ		%comm min	%comm max	%wall	funciones mpi (%mpi)	
openmp	4 nodos 4 tasks 48 thread/task	2.72243	4.76356	3.62	# MPI_Waitall # MPI_Init_thread # MPI_Barrier # MPI_Scan # MPI_Isend # MPI_Irecv # MPI_Bcast # MPI_Comm_split # MPI_Allgather # MPI_Allreduce # MPI_Wtime # MPI_Finalize # MPI_Comm_size # MPI_Comm_rank	94.01 4.55 0.62 0.38 0.23 0.13 0.06 0.02 0.00 0.00 0.00 0.00 0.00 0.00
	8 nodos 8 tasks 48 thread/task	2.78206	8.09617	6.84	# MPI_Waitall # MPI_Init_thread # MPI_Barrier # MPI_Scan # MPI_Isend # MPI_Irecv # MPI_Bcast # MPI_Comm_split # MPI_Allgather # MPI_Allreduce # MPI_Finalize # MPI_Wtime # MPI_Comm_size # MPI_Comm_rank	94.18 4.84 0.41 0.20 0.16 0.12 0.08 0.01 0.00 0.00 0.00 0.00 0.00 0.00
int.1	4 nodos 8 tasks 24 threads/task	0.67234	1.98598	1.46	# MPI_Waitall # MPI_Init_thread # MPI_Barrier # MPI_Irecv # MPI_Isend # MPI_Bcast # MPI_Scan # MPI_Comm_split # MPI_Allgather # MPI_Allreduce # MPI_Finalize # MPI_Wtime # MPI_Comm_size # MPI_Comm_rank	86.28 12.68 0.32 0.20 0.18 0.14 0.10 0.09 0.01 0.00 0.00 0.00 0.00 0.00
	8 nodos	1.52421	3.6266	2.69	# MPI_Waitall	79.79

	16 tasks 24 thread/task				# MPI_Init_thread # MPI_Barrier # MPI_Isend # MPI_Irecv # MPI_Scan # MPI_Bcast # MPI_Comm_split # MPI_Allgather # MPI_Allreduce # MPI_Finalize # MPI_Wtime # MPI_Comm_size # MPI_Comm_rank	18.78 0.37 0.24 0.23 0.22 0.19 0.17 0.01 0.00 0.00 0.00 0.00 0.00

Análisis OpenMP

BT-MZ

Los códigos que incluyen cláusulas y directivas OpenMp son: rhs.f (compute_rhs), x_solve.f, y_solve.f, z_solve.f, add.f exact_rhs.f, initialize.f, error.f.
Todas las partes paralelizadas tienen una estructura parecida. El tipo de paralelismo predominante es a nivel de bucles do, con schedule static y en muchos casos con bucles anidados con el uso de collapse(2). En algunos casos se utiliza nowait para evitar perder tiempo por sincronización al final de un bucle cuando no es necesario.
En algunos casos del programa error.f se utiliza la directiva atomic para evitar problemas de sincronización.

SP-MZ

Los códigos que incluyen cláusulas y directivas OpenMp son: add.f, exch_qbc.f, initialize.f, ninvr.f, pinvr.f, txinvr.f, tzetar.f, x.solve.f, y_solve.f, z_solve.f, exact_rhs.f, error.f, rhs.f.
Todas las partes paralelizadas tienen una estructura parecida. El tipo de paralelismo predominante es a nivel de bucles do, con schedule static y en muchos casos con bucles anidados con el uso de collapse(2). En algunos casos se utiliza nowait para evitar perder tiempo por sincronización al final de un bucle cuando no es necesario.
En algunos casos del programa error.f y exact_rhs.f se utiliza la directiva atomic para evitar problemas de sincronización.
En el programa rhs.f se utiliza a directiva #omp master, que indica que el bloque debe ser ejecutado por el “master thread”.

LU-MZ

Los códigos que incluyen cláusulas y directivas OpenMp son: blts.f, buts.f, jacu.f, setbv.f, erhs.f, lu.f, error.f, l2norm.f, exchange_1.f, rhs.f, pintgr.f, syncs.f.
El tipo de paralelismo predominante es a nivel de bucles do, con schedule static y en muchos casos con bucles anidados con el uso de collapse(2). En algunos casos se utiliza nowait para evitar perder tiempo por sincronización al final de un bucle cuando no es necesario. También es abundante el uso de la directiva atomic y barrier para evitar problemas de sincronización.
En los programas exchange_1.f y ssor.f se utiliza la directiva master. En el programa pintg.f se utiliza la cláusula reduction con la operación + para acumular resultados parciales.
El programa syncs.f utiliza la directiva #omp flush, que hace que la memoria del thread específico se adapte a la memoria global.

Código utilizado

---

config/suite.def
Copiado de config/suite.def.template, modificado para compilar los kernels de la clase D.

config/make.def
Copiado de config/make.def.template, modificado para tener las opciones de compilación adecuadas.

> *FC = mpiifort*
> *FFLAGS = -O3 -qopenmp -mt_mpi*
> *CC = mpiicc*
> *CFLAGS = -O3 -qopenmp -mt_mpi*

lu.sh, sp.sh, bt.sh
Scripts para ejecutar el código mediante sbatch, ejecuta cada programa 3 veces para poder calcular medias.

lu2.sh, sp2.sh, bt2.sh
Scripts para ejecutar el código mediante sbatch, modificado para crear un fichero con la información del análisis ipm, cada programa se ejecuta una vez.

adi.f, adi.f, lu.f
Código modificado para marcar regiones para el análisis ipm.