Comparação experimental dos algoritmos coletivos para o MPI_Allgather no OpenMPI

Wilton Jaciel Loch [1], **Guilherme Piêgas Koslovski** [1]

[1]Programa de Pós-Graduação em Computação Aplicada (PPGCA)
Universidade do Estado de Santa Catarina (UDESC) - Joinville, SC - Brasil

Resumo. *Uma das chamadas coletivas populares trata-se da MPI_Allgather, cuja implementação seleciona um algoritmo dentre os vários disponíveis com base nos parâmetros de execução e utilizando regras estáticas. O objetivo deste trabalho é avaliar experimentalmente as regras de seleção de algoritmo no OpenMPI 4.0.3 para esta chamada. Os resultados obtidos apontam que as escolhas são ineficientes para os cenários avaliados em grande parte dos testes.*

1. Introdução e metodologia experimental

As comunicações coletivas no *Message Passing Interface* (MPI) representam uma das principais formas de trocas de mensagem e dividem-se entre envios de um para muitos e muitos para muitos [MPI Forum 2015]. Para ambas as formas há uma multitude de algoritmos com desempenhos variáveis, cuja escolha depende dos parâmetros de execução. Experimentos foram realizados comparando os algoritmos disponíveis para a chamada coletiva `MPI_Allgather` na implementação OpenMPI 4.0.3, que no momento da escrita deste trabalho trata-se da versão de distribuição mais recente. Os algoritmos utilizados nesta chamada são: *linear*, *bruck*, *recursive doubling*, *ring*, *neighbor exchange* e *two proc*. O algoritmo *two proc* tem sua operação limitada apenas a dois processos, e portanto foi removido dos testes. De forma semelhante, o algoritmo *recursive doubling* opera apenas com quantidades de processos que são potências de dois, e portanto foi mantido apenas nos testes pertinentes. São utilizadas 62 e 128 como quantidades de processos (distribuídos ciclicamente) para representar casos gerais de valores pares e potências de dois, respectivamente. A quantidade de dados enviados é variada entre 32 *Bytes* e 512 *KiloBytes* para englobar blocos pequenos, médio e grandes, em multiplicações sucessivas por 4 devido ao limite de espaço. O tempo de execução é medido através de contadores próprios do MPI (`MPI_Wtime`) em aplicações *ad hoc*. Cada configuração de teste é repetida 50 vezes. Os experimentos foram realizados em 2 computadores homogêneos do aglomerado Cervino da Universidade de Neuchâtel (Unine). Ambos possuem 2 processadores Intel Xeon E5-2683 v4 (16 núcleos cada) e 128GB de memória. Ambas as máquinas são conectadas por uma rede Ethernet de 40Gbps.

2. Resultados

A Tabela 1 apresenta os resultados dos experimentos e é dividida em duas seções relativas às quantidades de processos. O tempo de cada algoritmo nas diferentes cargas é mostrado em dezenas de milissegundos e as células dos menores e maiores resultados de cada carga são coloridas em verde e vermelho, respectivamente. A linha intitulada OpenMPI representa as escolhas de algoritmo feitas pela implementação e os escolhidos para cada carga são destacados em negrito. Por fim, na mesma linha e em cada célula, uma flecha ascendente representa a escolha do melhor algoritmo enquanto uma descendente a escolha do pior.

	Tempo `MPI_Allgather` ($\times 10\mu s$)								
	Algoritmo	32B	128B	512B	2KB	8KB	32KB	128KB	512KB
62 processos	OpenMPI	$\uparrow 32 \pm 153$	$\uparrow 31 \pm 152$	$\uparrow 36 \pm 154$	246 ± 177	359 ± 174	1583 ± 176	6467 ± 4477	25573 ± 4736
	Linear	49 ± 288	67 ± 295	94 ± 293	206 ± 285	1729 ± 5040	2801 ± 4797	7728 ± 4053	23723 ± 4873
	Bruck (0-2)	$\mathbf{29 \pm 149}$	$\mathbf{32 \pm 153}$	$\mathbf{45 \pm 153}$	53 ± 153	100 ± 158	451 ± 149	3067 ± 216	12461 ± 1366
	Ring	306 ± 151	266 ± 176	242 ± 152	304 ± 166	451 ± 197	1388 ± 446	5738 ± 2920	29007 ± 8456
	Neighbor exchange (3-7)	129 ± 157	118 ± 165	134 ± 162	$\mathbf{242 \pm 161}$	$\mathbf{392 \pm 264}$	$\mathbf{1675 \pm 305}$	$\mathbf{6116 \pm 3775}$	$\mathbf{26317 \pm 6144}$
128 processos	OpenMPI	$\uparrow 67 \pm 418$	45 ± 208	799 ± 775	1052 ± 421	1831 ± 742	5994 ± 251	27294 ± 6882	104481 ± 16336
	Linear	92 ± 595	208 ± 721	166 ± 428	656 ± 744	4590 ± 10770	9665 ± 9677	35678 ± 16620	292064 ± 79559
	Bruck	209 ± 578	52 ± 201	93 ± 436	211 ± 548	716 ± 514	3462 ± 548	14270 ± 1270	52994 ± 6152
	Recursive doubling (0-2)	$\mathbf{52 \pm 323}$	$\mathbf{122 \pm 456}$	$\mathbf{104 \pm 574}$	74 ± 211	265 ± 301	1450 ± 403	5798 ± 401	25609 ± 3360
	Ring	669 ± 233	793 ± 578	684 ± 595	976 ± 714	1610 ± 622	6626 ± 3970	32383 ± 11268	147311 ± 25497
	Neighbor exchange (3-7)	347 ± 701	379 ± 736	507 ± 707	$\mathbf{574 \pm 719}$	$\mathbf{3952 \pm 6618}$	$\mathbf{7190 \pm 5379}$	$\mathbf{27648 \pm 7682}$	$\mathbf{101899 \pm 12278}$

Tabela 1. Tempos de execução da chamada `MPI_Allgather`.

Inicialmente, é válido destacar que o melhor desempenho alterna entre o *bruck* e o *recursive doubling* em todas as cargas, enquanto o pior desempenho alterna entre o *ring* e *linear*. Este comportamento é explicado pelos custos teóricos dos algoritmos, ambos *ring* e *neighbor exchange* tem complexidades lineares, *bruck* e *recursive doubling* possuem complexidades logarítmicas e o *linear* complexidade quadrática, justificando suas ascenções vertiginosas de tempo com o crescimento dos dados. Os algoritmos de complexidade linear são empregados pelo OpenMPI porque embora o *recursive doubling* e *bruck* possuam complexidades teóricas logarítmicas, na prática seus padrões de comunicação são difusos e para o último são ainda necessárias cópias e movimentações de memória em sua operação. Estes fatores tornam seus tempos de execução maiores que os competidores com o crescimento da infraestrutura, processos e dados. Portanto, a realização dos testes com grandezas maiores para as variáveis e mais máquinas interligadas tenderia a uma superação dos algoritmos lineares sobre o *bruck* e o *recursive doubling*. Por fim, é possível verificar que as escolhas determinísticas do OpenMPI são ineficientes para o ambiente de testes empregado. Com 62 processos, em apenas 3 das 8 cargas o algoritmo escolhido era de fato o melhor, ao passo que um algoritmo sub-ótimo foi escolhido nos casos restantes, resultando em escolhas ineficientes em 5 dos 8 testes. Com 128 processos os resultados são ainda piores, já que em apenas um dos testes o melhor algoritmo foi escolhido e nos restantes foram tomados algoritmos sub-ótimos, implicando em escolhas ineficientes em 7 dos 8 casos.

3. Considerações Finais

Os resultados dos testes demonstram que há de fato ineficiência na seleção de algoritmos para a operação coletiva `MPI_Allgather` e sugerem que problemas semelhantes podem também existir para outras chamadas. De forma geral, este trabalho serve de justificativa e ponto de partida para o desenvolvimento de ferramentas de auto ajuste dos parâmetros para seleção de algoritmos. As observações preconizam o desenvolvimento de novos algoritmos que tenham complexidade logarítmica, comunicação local e não façam cópias e movimentações de memória, unindo as vantagens de diferentes algoritmos já existentes. Como trabalhos futuros se indica a execução de testes maiores e o desenvolvimento de algoritmos mais eficientes e ferramentas inteligentes para a escolha.

Agradecimentos: Este trabalho foi desenvolvido com financiamento da FAPESC e apoio da Unine através do acesso à infraestrutura de testes.

Referências

MPI Forum (2015). MPI: A Message-Passing Interface Standard.