Analysis

Block Size 16:

Generally, block size 16 performs worse than the non-blocked version for larger matrix sizes (2048 and 4096), indicating poor utilisation of the cache and significant overhead. However, for matrix size 1024, the blocked version often outperforms the non-blocked one, suggesting some cache benefit for smaller matrices.

Block Size 32:

A block size of 32 shows mixed results. For matrix size 1024, there is a consistent speedup, indicating better cache utilisation than block size 16. However, for larger matrix sizes (2048 and 4096), the speedup is inconsistent, suggesting that block size 32 still doesn't efficiently utilise the cache for larger matrices.

Block Size 64:

With block size 64, there's an observable improvement in speedup across all matrix sizes, particularly for 2048 and 4096. This suggests that block size 64 is more effectively utilising the cache, likely fitting better within the L1 and tapping into the L2 cache.

Block Size 128:

This block size generally offers higher speedup values across all matrix sizes. The results indicate effective utilisation of the L2 cache, aligning with the size of the L2 cache on my machine. The block likely fits within the L2 cache, reducing frequent memory accesses.

Block Size 256:

Block size 256 demonstrates significant speedup, especially for larger matrix sizes. This size effectively utilises the L2 cache, offering an optimal balance between cache hits and the overhead of managing larger blocks.

Block Size 512:

With block size 512, there's a notable speedup for larger matrix sizes, suggesting effective cache utilisation. However, for matrix size 1024, the speedup is not consistent, indicating that this block size may start to have diminishing returns for smaller matrices.

Block Size 1028:

This block size generally underperforms for matrix size 1024 but shows improvements for larger matrix sizes. It seems that the overhead of managing such large blocks might outweigh the caching benefits for smaller matrices.

Overall Analysis:

→ Smaller block sizes (16 and 32) often underperform compared to larger block sizes, especially for larger matrix sizes, likely due to overheads and poor cache utilisation.

→ Medium block sizes (64, 128) show improved performance, indicating better cache utilisation. The speedup for larger matrices is particularly noticeable.

→ Larger block sizes (256, 512) generally offer significant speedup across different matrix sizes, likely making efficient use of the L2 cache. However, there might be some overhead for smaller matrices.

→ Very large block size (1028) seems to have diminishing returns, particularly for smaller matrix sizes, due to the overhead of managing large blocks.

In conclusion, the block size significantly impacts cache utilisation and execution time. The optimal block size depends on both the cache size and the overhead of managing blocks. For my specific machine configuration, block sizes around 256 to 512 might provide the best performance for blocked matrix multiplication, especially for larger matrices. However, the overhead for smaller matrices and larger block sizes should be considered.

Cache sizes:
hw.l1icachesize: 131072
hw.l1dcachesize: 65536
hw.l2cachesize: 4194304

Experimental data:

Block size	Matrix size	Blocked Time	Non-Blocked Time	Speedup
16	1024	0.046239s	0.018986s	0.410606
16	2048	0.384312s	0.143710s	0.373941
16	4096	3.130451s	2.508537s	0.801334
32	1024	0.028290s	0.021554s	0.761895
32	2048	0.277467s	0.456042s	1.643590
32	4096	3.815504s	3.107325s	0.814394
64	1024	0.030180s	0.017693s	0.586249
64	2048	0.219597s	0.527424s	2.401781
64	4096	2.907177s	2.969993s	1.021607
128	1024	0.042887s	0.056455s	1.316366
128	2048	0.223838s	0.514336s	2.297805
128	4096	2.781985s	3.262335s	1.172664
256	1024	0.023403s	0.033783s	1.443533
256	2048	0.212590s	0.488914s	2.299798
256	4096	2.549388s	3.116296s	1.222370

Block size	Matrix size	Blocked Time	Non-Blocked Time	Speedup
16	1024	0.062537s	0.017852s	0.285463
16	2048	0.642908s	0.469193s	0.729798
16	4096	6.187844s	3.061988s	0.494839
32	1024	0.036295s	0.096505s	2.658906
32	2048	0.367765s	0.491850s	1.337403
32	4096	3.418517s	3.410878s	0.997765
64	1024	0.022508s	0.018598s	0.826284
64	2048	0.209081s	0.520263s	2.488332
64	4096	2.851040s	3.237514s	1.135555
128	1024	0.036945s	0.052921s	1.432427
128	2048	0.247224s	0.595366s	2.408205
128	4096	2.971795s	3.101402s	1.043612
256	1024	0.043219s	0.085110s	1.969273
256	2048	0.334556s	0.664300s	1.985617
256	4096	3.057045s	4.975457s	1.627538

Block size	Matrix size	Blocked Time	Non-Blocked Time	Speedup
16	1024	0.094477s	0.071551s	0.757338
16	2048	0.700460s	0.683929s	0.976400
16	4096	6.236812s	3.137143s	0.503004
32	1024	0.033398s	0.054421s	1.629469
32	2048	0.292032s	0.511552s	1.751698
32	4096	3.243625s	3.090570s	0.952814
64	1024	0.029919s	0.017607s	0.588489
64	2048	0.159700s	0.513904s	3.217934
64	4096	2.703403s	3.270794s	1.209880
128	1024	0.051960s	0.091520s	1.761355
128	2048	0.243203s	0.493615s	2.029642
128	4096	2.767221s	3.209778s	1.159928
256	1024	0.018417s	0.018157s	0.985883
256	2048	0.226743s	0.526536s	2.322171
256	4096	2.991613s	3.413341s	1.140970