1. Optimization Result

	Baseline	1024	768	512	256	128
Speed up	1	2.839513	2.402532	2.400022	2.379537	2.356146

2. Analysis of Results and Strategy

For a size of 1024, a speed increase of about 2.8 times was achieved, while for the other sizes, an increase between 2.3 and 2.4 times was observed. The larger the size, the more complex the calculations and cache management become, which makes the difference between optimized and non-optimized code more pronounced.

As part of the optimization strategy, unnecessary declarations and address allocations in the code were removed. Parts of the code that could function without address allocations or unnecessary variables were minimized to make efficient use of the cache.

Secondly, calculations were simplified. Efforts were made to minimize multiplication operations, and calculations within loops were reduced as much as possible.

Thirdly, loop unrolling was performed. To prevent unnecessary repetitions in loops, short loops were directly implemented, minimizing calculations for efficiency.

Fourthly, data size was reduced. When declaring variables, smaller data types were chosen whenever possible. Additionally, efforts were made to shorten the code length to reduce instruction cache usage, aiming for more concise and logical code.

Lastly, efforts were made to ensure that array accesses were as sequential as possible. Loops were modified to access arrays sequentially, thereby ensuring spatial locality.

Furthermore, simple comparisons such as == were preferred over more complex comparisons like > or >=, and inline techniques were used wherever possible to further optimize performance.