BLAS3

- BLAS2 was motivated by efficient vector register/memory port usage

- Reuse of data is fairly low: $\mu_{\text{min}} \geq 1/2$

- Unsuitable for a hierarchical memory architecture: large shared main memory and a much smaller local memory or cache that may or may not be shared by more than one processor

- BLAS3 moves up one level of complexity to matrix-matrix operations

- subsumes the previous two levels of the hierarchy
We will assume that we are on a vector processor which has a large main memory with size $M$ and smaller local memories or caches with size $CS << M$. The cache or local memory is assumed to have the necessary bandwidth to keep up with the processors' vector operations.
BLAS3:

- '85 Calahan - block LU Cray 2, Bischof and Van Loan - block Householder, Libraries: C-SRD on multivector processor, Gustavson and IBM on IBM 3090 (begun in 1984)
- many papers in latter part of '86 and '87
- early '87 Dongarra et. al. - standardization effort
- LAPACK
- matrix – matrix operations
- $O(n^3)$ operations, $O(n^2)$ data
- allows efficient management of vector registers, cache, and memory due to data locality
- 3-D parallelism
The basic BLAS3 computation is

$$C \leftarrow C + AB,$$

where $C$, $A$ and $B$ are $n_1 \times n_3$, $n_1 \times n_2$, and $n_2 \times n_3$ matrices, respectively. Clearly, this primitive subsumes the rank-1 update, $(n_2 = 1)$, and matrix-vector multiplication, $(n_3 = 1)$, BLAS2 primitives. In block algorithms, it is most often used as a rank-$\omega$ update $(n_2 = \omega \ll n_1, n_3)$ or a matrix multiplied by several vectors $(n_3 = \omega \ll n_1, n_2)$.

The basic scalar computation can be expressed as

$$\gamma_{s,r} \leftarrow \gamma_{s,r} + \sum_{t=1}^{n_2} \alpha_{s,t} \beta_{t,r}$$

for $1 \leq s \leq n_1$ and $1 \leq r \leq n_3$, where $\gamma_{s,r}$, $\alpha_{s,t}$, and $\beta_{t,r}$ denote the elements of $C$, $A$ respectively $B$. 
Three Basic Approaches

There are three basic generic approaches to performing these computations: *inner*, *middle*, and *outer* product forms.

inner product form:

\[ \gamma_{s,r} \leftarrow \gamma_{s,r} + a_s^T b_r \]

where \( a_s^T = e_s^T A \) and \( b_r = B e_r \).

We have \( n_1 n_3 \) inner products of length \( n_2 \), i.e., multiple independent BLAS1 primitives.
middle product:

\[ c_r \leftarrow c_r + Ab_r \]

where \( c_r = Ce_r \) and \( b_r = Be_r \). We have \( n_3 \) matrix-vector products of size \( n_1 \times n_2 \), i.e. multiple independent BLAS2 reduction primitives
\[ C \leftarrow C + \sum_{t=1}^{n_2} a_t b^T_t \]

where \( a_t = A e_t \) and \( b^T_t = e^T_t B \).

We have \( n_2 \) rank-1 updates of size \( n_1 \times n_3 \), i.e., multiple BLAS2 nonreduction primitives. Note however that there are dependences between the rank-1 updates since the result is a sum of the results of the primitives.

It is also possible to define other forms which organize the computation by diagonals. Strassen developed a version such that the total number of operations is less than \( 2n_1 n_2 n_3 \).
Basic Parameters

It follows immediately from the inner product form that

\[ \Omega = 2n_1n_2n_3 \]
\[ \delta = n_1n_2 + n_1n_3 + n_2n_3 \]
\[ \Theta_{\text{min}} = n_1n_2 + 2n_1n_3 + n_2n_3 \]
\[ \pi_{\text{max}} = n_1n_2n_3 \]
\[ \pi_{\text{ave}} = \frac{(2n_1n_2n_3)}{\left(\log(n_2) + 2\right)} \]
\[ \mu_{\text{min}} = \frac{1}{n_2} + \frac{1}{2n_1} + \frac{1}{2n_3} \]
Block Generalizations

Each of the approaches have generalizations which work on submatrices rather than elements. We will assume that all \( p \) functional units or processors share the cache. (Not typical of newer parallel processors.)

Partition \( A, B, \) and \( C \) into submatrices:

\[ C_{ij}: m_1 \times m_3, A_{ik}: m_1 \times m_2, B_{kj}: m_2 \times m_3 \]

Perform block form:

\[
\begin{align*}
\text{do } i = 1, k_1 \\
& \quad \text{do } k = 1, k_2 \\
& \quad \quad \text{do } j = 1, k_3 \\
& \quad \quad \quad C_{ij} = C_{ij} + A_{ik} \times B_{kj} \\
& \quad \text{end do} \\
& \text{end do}
\end{align*}
\]

where \( n_1 = k_1 m_1, n_2 = k_2 m_2, \) and \( n_3 = k_3 m_3, \) and \( k_1, k_2, \) and \( k_3 \) are assumed to be positive integers for simplicity.

All \( p \) processors cooperate on one block operation \( C_{ij} = C_{ij} + A_{ik} \times B_{kj} \) using the

middle_product:

\[
\begin{align*}
\text{do concurrently on VFPUs' } r = 1, m_3 \\
& \quad c_{*,r} = c_{*,r} + A_{ik} b_{*,r} \\
& \text{end do}
\end{align*}
\]

\( b \) and \( c \) now represent vectors of size \( m_2 \) and \( m_1 \) respectively
Choosing Parameters

Goal: develop a strategy, applicable at the kernel and algorithm level, to aid in choosing algorithm parameter- s based on a simple model and empirical observations of system behavior.

\[ T = T_a + \Delta_l = \tau_a n_a + \tau m_l \]

- \( T_a \) is the arithmetic time using one level of fast memory
- \( \Delta_l \) is the degradation due to the finite cache
- \( \tau_l \) and \( \tau_a \) are the average times for a data transfer and a computation respectively
- \( n_l \) and \( n_a \) are the number of data transfers from memory to cache and the number of operations respectively
- we define \( \lambda = \tau_l / \tau_a \) and \( \mu = n_l / n_a \)
Global optimization is too difficult so we decouple the terms and minimize $T_a$ and $\Delta l/T_a$ separately. The algorithm paramters are then taken from the intersection.
$T_a$ is reduced by considering:

- performance of kernel in terms of block sizes ignoring effect of cache

- instruction timings and simple models of vector and multi-vector processors, e.g., Hockney’s and extensions

- management of register-memory hierarchy (for our example this is an analysis of executing a matrix-vector product on each VFPU of using all of the techniques we applied to analyzing the BLAS2)

- For registers of length 32 we have, $m_1 = 32i$ for some $i$ or large, $m_2 > 16$ or so to accumulate enough results of triads in registers before storing to efficiently use the memory port, $m_3 = 8l$ or large for load balancing across the CEs.

$\Delta_l/T_a = \lambda \mu$ is reduced by considering:

- $\mu$ as a function of algorithm parameters, assuming perfect local memory or cache behavior, given a bound on $\lambda$ based on hardware considerations and empirical data

- techniques, based on empirical data of various primitives with operands from memory and cache, which mitigate certain performance degrading effects due to actual cache or local memory implementations, e.g., the direct map from memory to cache on machines like the Alliant causes difficulties.
Data transfer analysis

To be consistent with the discussion in the reference distributed we will define $\mu$ in terms of loads from memory to cache only. (If $\lambda$ is bounded conservatively this is not that significant of a simplification)

From the block algorithm above it is clear that $A_{ik}$ is what is assumed to be kept in cache over the entire $j$-loop. We will assume that $C_{ij}$ and $B_{kj}$ must be reloaded on every iteration. Therefore the number of loads from memory to cache is

$$k_1k_2(m_1m_2 + k_3(m_1m_3 + m_2m_3))$$

Since $k_i = n_i/m_i$,

$$\mu = \frac{1}{2} \left( \frac{1}{m_1} + \frac{1}{m_2} \right) + \frac{1}{2n_3}$$
The optimization of data loading is equivalent to:

\[
\min n_1 n_2 + n_1 n_2 n_3 \left( \frac{1}{m_1} + \frac{1}{m_2} \right)
\]

subject to

\[
m_2(m_1 + p) \leq CS, \quad 1 \leq m_1 \leq n_1, \quad 1 \leq m_2 \leq n_2
\]

This has the solution:

1. The value of \( m_3 \) is arbitrary and taken as \( n_3 \) for maximum parallelism.

2. If \( n_2(n_1 + p) \leq CS \), \( m_i = n_i \), for \( i = 1, 2 \) (Regime 1).

3. If \( n_2(n_1 + p) > CS \), and \( n_2 \leq CS(CS^{1/2} + p)^{-1} \) (Regime 2)

\[
m_1 = \frac{CS}{n_2} - p \quad m_2 = n_2.
\]

4. If \( n_2(n_1 + p) > CS \), and \( n_1 \leq CS^{-1/2} \) (Regime 3)

\[
m_1 = n_1 \quad m_2 = \frac{CS}{n_1 + p}.
\]

5. If \( n_2(n_1 + p) > CS \), and \( n_1 > CS^{-1/2} \) and \( n_2 > CS(CS^{1/2} + p)^{-1} \) (Regime 4)

\[
m_1 = CS^{1/2} \quad m_2 = \frac{CS}{CS^{1/2} + p}.
\]
Solution Regimes

where

\[
\mu = \frac{1}{2} \left( \frac{1}{m_1} + \frac{1}{m_2} + \frac{1}{n_3} \right),
\]

\[A = CS/(CS^{1/2} + p)^{-1}\] and \[B = CS^{1/2}].

- Regime (1) \( m_i = n_i \), \( \mu \) is minimum but not necessarily acceptable.

- Regimes (2) and (3) \( \mu = O(1/\omega) \), \( \omega = \min(n_1, n_2) \).

- Regime (4) \( m_1 \approx m_2 \approx CS^{1/2} \) and \( \mu \approx CS^{-1/2} \).
Regime 2 is a typical BLAS3 primitive used in block algorithms, the rank-$\omega$ update, i.e., $n_1$ and $n_3$ large and $n_2 = \omega$ which is much smaller.

Note the hyperbolic behavior of $\mu$. A little bit of blocking helps a lot but improvement rate diminishes as larger blocks are used in the block algorithm ($\omega$ is often a block size in a block algorithm for computations such as $LU$ factorization) This trend is apparent from the performance graphs shown later for the matrix-matrix product and the rank-$\omega$ update.

Also note the limit of $\mu \approx 1/\sqrt{CS}$ Therefore there is a diminishing return effect on increasing the cache size of the machine. If $CS$ is increased by a factor of $\alpha$ then $\mu$ reduces by a factor of only $1/\sqrt{\alpha}$
Typical partitionings

All dimensions large:

$$\left( \begin{array}{c} C_1 \\ \vdots \\ C_k \end{array} \right) \leftarrow \left( \begin{array}{c} C_1 \\ \vdots \\ C_k \end{array} \right) \pm \left( \begin{array}{ccc} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{k1} & \cdots & A_{km} \end{array} \right) \left( \begin{array}{c} B_1 \\ \vdots \\ B_m \end{array} \right),$$

Rank-ω update

$$\left( \begin{array}{c} C_1 \\ \vdots \\ C_k \end{array} \right) \leftarrow \left( \begin{array}{c} C_1 \\ \vdots \\ C_k \end{array} \right) \pm \left( \begin{array}{c} A_1 \\ \vdots \\ A_k \end{array} \right) B,$$  \hspace{1cm} (1)