

# Weekly Research Progress Report

Student: Xing Cong

Date: 01/09/2023 - 05/09/2023

## List of accomplishments this week:

Read paper *Cache-aware Sparse Patterns for the Factorized Sparse Approximate Inverse Preconditioner*

## Paper summary:

Name :

Cache-aware Sparse Patterns for the Factorized Sparse Approximate Inverse Preconditioner.

## Introduction:

This paper proposes and evaluates an approach to extend FSAI sparse patterns based on two fundamental concepts: First, a cache-aware algorithm to extend sparse patterns, which reduces CG iteration count while keeping the cost per iteration low. Such cache-aware optimization relies on low-level aspects of the cache hierarchy architecture . Second, an approach to filter out the smallest entries of the cache-aware FSAI pattern extension without degrading its convergence properties.

As a result, these methods obtain very significant reductions in terms of average solution time ranging between 12.94% and 22.85% on three different architectures - Intel Skylake, POWER9 and A64FX - over a set of 72 test matrices.

## Motivation :

Because of many parallel computing and matrix multiplication methods are used in HYCOM, so we attempt to be inspired by the optimization research on the conjugate gradient method pre-processor in this paper, so that we can optimism the computational speed and efficiency of HYCOM.

**Work summary**

The notes I took have already been included in the appendix.

**Next**

Continuing the research on relevant optimization methods through literature review.

Attempt to deploy HYCOM on a computer for execution.

# Cache-aware Sparse Patterns for the Factorized Sparse Approximate Inverse Preconditioner

论文原文地址：“Cache-aware Sparse Patterns for the Factorized Sparse Approximate Inverse Preconditioner” ([pdf](#))

## Abstract

对于求解该线性方程  $Ax = b$ ，当  $A$  矩阵是对称且正定时，通常的解决办法是使用：共轭梯度法（Conjugate Gradient），可以简称为CG。

而CG的求解效率和  $A$  矩阵的特征值分布有关，为了加速该方程的求解，可以考虑对  $A$  进行线性变换，使得矩阵  $A$  的特征值分布的更为密集，这里就引入了一个方法，就是使用某种矩阵对  $A$  进行线性变换，从而得到更为密集的特征值分布。

“Part of its effectiveness relies on finding a suitable pre-conditioner that accelerates its convergence.” ([Laut 等, 2021, p. 1](#)) ([pdf](#)) CG的效率部分取决于找到一个加速其收敛的适当预处理器。也就是这个矩阵  $M$ 。

★寻找这个矩阵的方法：**Factorized Sparse Approximate Inverse (FSAI) pre-conditioners** are a prominent and easily parallelizable option.，这个和利用  $A^{-1}$  进行变换的方法类似，但在计算机上运行更为快速也便于进行并行计算。因为这个不需要进行求逆运算。

★这篇文章主要贡献：“we introduce **complementary architecture-aware criteria** to increase the numerical effectiveness of the **pre-conditioner**. In particular, **we define cache-aware pattern extensions** that do not trigger additional cache misses when accessing vector  $x$  in the  $y = Ax$  Sparse Matrix-Vector (SpMV) kernel.” ([Laut 等, 2021, p. 1](#)) ([pdf](#)) ”我们引入互补的面向架构的标

准，以增加预条件器的数值效果。特别是我们定义了缓存感知的模式扩展，在使用矢量 $x$ 访问 $y = Ax$ 稀疏矩阵-向量(SpMV)核时不触发附加的缓存未命中。

Sparse Matrix-Vector (SpMV) kernel 是指执行稀疏矩阵向量乘法的计算核心。它是一种常见的计算机科学和数值计算中的操作，用于将稀疏矩阵与向量相乘。SpMV kernel 的目标是高效地执行这种乘法运算，以便在处理大规模稀疏矩阵时能够提供较快的计算速度和较低的内存占用。SpMV kernel 的实现方式可以根据具体的硬件架构和优化目标而有所不同。



- introduce **complementary architecture-aware criteria**
- define **cache-aware pattern extensions**



★论文的改进效果：“As a result, we obtain very significant reductions in terms of average solution time ranging between 12.94% and 22.85% on three different architectures - Intel Skylake, POWER9 and A64FX - over a set of 72 test matrices.” (pdf) 因此，我们在三种不同架构（Intel Skylake、POWER9和A64FX）上对72个测试矩阵进行了评估，得出了平均解决时间的显著减少，范围在12.94%至22.85%之间。

## Introduction

在求解线性方程时

- “Direct methods like the sparse LU factorizations are not useful in this context due to their memory requirements and the significant number of steps they take.” (Laut 等, 2021, p. 1) abc 直接的方法，如稀疏LU分解，在这种情况下不太有用，因为它们需要大量的内存，并且需要相当多的步骤。 abc
- “Thus, iterative methods are the best option and, in particular, Krylov methods are very commonly applied due to their excellent convergence properties.” (Laut 等, 2021, p. 1) abc 因此，迭代方法是最佳选择，特别是由于其优异的收敛性质，Krylov方法非常常用。 abc

★特别的：当线性方程  $Ax = b$  中的  $A$  矩阵为对称正定矩阵时，“When dealing with symmetric and positive definite matrices a popular Krylov method, Conjugate Gradient (CG), is typically applied.” (Laut 等, 2021, p. 1)  当处理对称和正定矩阵时，通常会使用一种流行的克里洛夫方法——**共轭梯度法（CG）**。

“The performance of the SpMV is significantly influenced by irregular memory access patterns on  $x$  driven by the locations of the sparse matrix non-zero coefficients.” ([Laut 等, 2021, p. 1](#)) ([pdf](#))  稀疏矩阵向量乘法（SpMV）的性能受**稀疏矩阵非零系数的位置**所驱动，进而显著受到在  $x$  上的不规则内存访问模式影响。

而CG的求解效率和  $A$  矩阵的特征值分布有关，为了加速该方程的求解，可以考虑对  $A$  进行线性变换，使得矩阵  $A$  的特征值分布的更为密集，这里就引入了一个方法，就是使用某种矩阵对  $A$  进行线性变换，从而得到更为密集的特征值分布。

除了每个单独核心的性能特性外，另一个强烈影响CG解线性方程组容量的因素是矩阵  $A$  的条件数。

在这个背景下，**预条件器通常用于改善CG的收敛性**。从简单的Block-Jacobi [34] 到复杂的多网格技术[18]，已提出大量旨在实现有效预处理的方法。

稀疏近似逆(SAI)预处理器包括一种近似的逆矩阵  $M \approx A^{-1}$ ，并受到某种稀疏模式的限制[11, 12]。

★随后，求解等价和更好条件的系统  $MAx = Mb$ 。

实际上，应用SAI预处理器包括使用额外的SpMV核，使其高度并行化。在CG的上下文中，对称正定问题应用因式分解稀疏近似逆(FSAI)，这意味着通过因式分解  $G^T G$  来逼近  $A^{-1}$ ，而不是单一矩阵。

★FSAI的一个非常重要的方面是其**相应稀疏模式的定义**。尽管目前最先进的解决方案只考虑数值方面，但本文证明，在定义FSAI稀疏模式时，低层架构相关的概

念也应该被考虑进去。“While state-of-the-art solutions define this pattern by exclusively taking into account numerical considerations, we demonstrate in this paper that low-level architecture-aware concepts should also be taken into account when defining the FSAI sparse pattern.”(pdf) 当今最先进的解决方案仅通过考虑数字因素来定义这一模式，而我们在本文中展示，当定义FSAI稀疏模式时，还应考虑低级架构意识的概念。

## 本文的贡献

本文提出并评估了一种扩展基于FSAI的稀疏模式的方法。

该方法基于两个基本概念：

- 首先，一种缓存感知算法用于扩展稀疏模式，从而降低CG迭代次数，同时保持每次迭代的成本较低。
  - 这种缓存感知优化依赖于缓存层次体系结构的底层细节，如索引机制或虚拟内存管理方法。
- 其次，一种过滤缓存感知FSAI模式扩展中最小条目的方法，而不降低其收敛性能。

本文相对于现有技术的贡献如下：

1. We propose a technique to obtain cache-friendly FSAI sparse patterns.
  - By considering some low-level aspects of the cache hierarchy architecture, **our algorithm is able to extend sparsity patterns in a way the number of iterations is reduced while the cost per iteration remains low enough to increase performance.**
  - Our approach is architecture independent as it just requires the cache line size as architecture input.
2. We propose a robust approach to **filter out small entries of the inverse approximation** without degrading the numerical properties of the FSAI pre-conditioner.

3. We **evaluate our proposals via an extensive evaluation campaign considering 72 matrices of the SuiteSparse Collection** [13] that fit in the available memory resources of a single node.
- Our experiments consider three high-end systems: a 48-core Skylake machine, a 40-cores POWER9, and a 48-cores A64FX.
  - In Skylake, our approach reduces timeto-solution by 15.02% on average.
  - In POWER9 and A64FX, these improvements are 12.94% and 22.85%, respectively.

## Background



### Conjugate Gradient (CG)

### Sparse Approximate Inverse Pre-conditioner (SAI)

稀疏近似逆 (Sparse Approximate Inverse, SAI) 预条件器基于逆矩阵中有许多可以忽略的小元素的假设, 只保留最大的元素, 并因此达到稀疏逼近的有效性。

在SAI方法的设置过程中, 寻找一个满足固定稀疏模式S的逆矩阵的近似  $M \approx A^{-1}$ 。考虑等效但条件更好的系统  $MAx = Mb$ 。

“When dealing with symmetric and positive definite problems, to preserve the system symmetry, the Factorized Sparse Approximate Inverse (FSAI) preconditioner is applied and  $A^{-1}$  is approximated by a factorization  $G^T G$  instead of a single matrix  $M$ , which means that two SpMV products are necessary instead of one.  $G$  is a sparse lower triangular matrix approximating the inverse of the Cholesky factor,  $L$ , of  $A$ .” (Laut 等, 2021, p. 2)

 当处理对称且正定问题时, 为了保持系统的对称性, 我们采用了Factorized Sparse Approximate Inverse (FSAI) 预处理器, 并且用★因式分解  $G^T G$  来逼近  $A^{-1}$  ★, 而不是单独的矩阵M, 这意味着需要两个SpMV乘积而不是一个。G是一个稀疏的下三角矩阵, 近似表示**A**的**Cholesky**分解因子**L**的逆。  $G \approx L^{-1}$  

## Cholesky分解因子

Cholesky 分解因子 (Cholesky factor) 是指将一个正定对称矩阵分解成一个下三角矩阵和其转置的乘积的过程。

$$A = LL^T$$

这个下三角矩阵被称为 Cholesky 分解因子。

$$\begin{pmatrix} 4 & 12 & -16 \\ 12 & 37 & -43 \\ -16 & -43 & 98 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 0 \\ 6 & 1 & 0 \\ -8 & 5 & 3 \end{pmatrix} \begin{pmatrix} 2 & 6 & -8 \\ 0 & 1 & 5 \\ 0 & 0 & 3 \end{pmatrix}$$

Cholesky 分解因子的计算可以用于求解线性方程组、计算矩阵的逆以及进行随机数生成等各种数值计算任务。

由于 **Cholesky 分解** 只需要计算下三角矩阵的元素，相对于 **LU 分解** 等其他矩阵分解方法，Cholesky 分解具有更高的计算效率和数值稳定性。

注意：

$$(AB)^{-1} = B^{-1}A^{-1}$$

所以， $A = LL^T$ ,  $A^{-1} = G^T G$  其中  $G \approx L^{-1}$  的推导过程如下：



$$\begin{aligned} A &= LL^T \\ A^{-1} &= (LL^T)^{-1} \\ A^{-1} &= (L^T)^{-1} L^{-1} \\ A^{-1} &= (L^{-1})^T L^{-1} \\ \text{if } L^{-1} &= G \text{ to :} \\ A^{-1} &= (G)^T G \end{aligned}$$



如何寻找G:

$$\min_{G \in S} \|I - GL\|_F^2$$

其中,  $A = LL^T$  和  $G \approx L^{-1}$

“We apply state-of-the-art techniques [11] to find  $G$  without explicitly evaluating  $L$ , i. e., only using the initial matrix  $A$ .” (Laut 等, 2021, p. 2)  我们使用最先进的技术[11]来找到没有明确评估L的G, 即只使用初始矩阵A。 



---

**Algorithm 1** FSAI,  $G^T G \approx A^{-1}$ 

---

- 1: Threshold  $A$  to produce  $\tilde{A}$ .
  - 2: Compute the pattern  $\tilde{A}^N$ , and let the pattern of  $G$  be the lower triangular part of the pattern of  $\tilde{A}^N$ .
  - 3: Compute the nonzero entries in  $G$  by solving the Frobenius minimization problem.
  - 4: Drop small entries in new  $G$  and rescale.
- 

## Accelerating FSAI

“This section describes a high-level view of our cache-aware sparse pattern extension strategy to boost the FSAI performance.” ([Laut 等, 2021, p. 2](#)) ([pdf](#))   
本节描述了我们的[高级缓存感知稀疏模式扩展策略](#), 以提高FSAI的性能。 

---

**Algorithm 2** FSAI,  $G^T G \approx A^{-1}$  with pattern extension and precalculation

---



- 1: Threshold  $A$  to produce  $\tilde{A}$ .
  - 2: Compute the pattern  $\tilde{A}^N$ , and let the pattern,  $S$ , of  $G$  be the lower triangular part of the pattern of  $\tilde{A}^N$ .
  - 3: **Compute cache-friendly extension of the pattern of  $G$ ,  $S_{ext}$ .**
  - 4: **Precalculate an approximation  $\tilde{G}$  of the preconditioner, and filter out entries of  $S_{ext}$  according to its values.**
  - 5: Calculate  $G$  on the sparse pattern obtained from the previous step.
- 

算法2显示了通过添加一个步骤，即增加**the cache-friendly extension of the sparse pattern**(稀疏模式的缓存友好扩展)，并通过一个更复杂的过滤处理应用于**G**的近似预计算结果，来替换G的筛选和重新缩放步骤的FSAI的重新制定。

- **The step** added in line 3 **extends the sparse pattern** of  $G$  considering architecture-aware criteria **to add additional entries** that reduce the CG iteration count while incurring a minimal overhead in terms of iteration cost.
  - Note that we propose an extension of the sparse pattern, therefore **the set of matrices** considered in the Frobenius minimization problem of Equation 3 **increases**. Consequently, **the new inverse approximation is more accurate**.
  - 在第四部分进行解释
- **The step** added in line 4 **replaces the filtration** that Algorithm 1 performs after the computation of  $G$ .
  - In this new filtration strategy, entries of the sparse pattern are filtered out based on an approximate pre-calculation of  $G$ .
  - 在第五部分进行解释

# Cache-Friendly Fill-in

“In this section we propose a **cache-friendly fill-in approach** to **extend the sparse pattern of the FSAI pre-conditioner**. We propose **architecture-aware techniques** to extend the sparse pattern in a way that we achieve significant reductions in terms of iteration count while minimizing the iteration cost overhead. In particular, we propose a method to extend the FSAI sparse pattern without increasing the number of cache misses.” ([Laut 等, 2021, p. 3](#)) ([pdf](#))

 在本节中，我们提出了一种缓存友好的填充方法来扩展FSAI预处理器的稀疏模式。我们提出了一种架构感知技术，以在减少迭代计数的同时最小化迭代成本开销的方式来扩展稀疏模式。具体而言，我们提出了一种在不增加缓存未命中次数的情况下扩展FSAI稀疏模式的方法。

Since FSAI is applied via the SpMV kernel  $y = Ax$ , we must consider the **access patterns** for all the involved data structures containing  $y$ ,  $A$ , and  $x$ .

- 访问A矩阵：
  - Assuming that we traverse the sparse matrix  $A$  **in row order** and that we **store it using the Compressed Sparse Row (CSR) format**, the accesses on the data structures containing  $A$  display a very simple stride-1 pattern. Since **this pattern is easily predictable** by hardware pre-fetchers, there is some flexibility for extending  $A$  without suffering a prohibitive performance penalty.
- 对于 $y$ :
  - 和访问A类似
- 对于 $x$ :
  - The most problematic accesses are those coming from accesses to **vector  $x$** , which follow a random pattern and thus **can not be easily predicted by the pre-fetcher**.

# Cache Alignment of Vector $x$

Our approach drives the extension of the FSAI sparse pattern by taking into account **the cache alignment of vector  $x$** .

“In other words, the idea is to add new coefficients in  $A$  that fully exploit the spatial locality on memory accesses to vector  $x$ .” ([Laut 等, 2021, p. 3](#)) ([pdf](#))

换句话说，意思是在矩阵 $A$ 中添加新的系数，充分利用对向量 $x$ 的内存访问的空间局部性。该方法依赖于通过使用相应的虚拟地址来确定缓存行中存储的向量元素 $x_i$ 的相对位置

我们可以通过虚拟地址的偏移位来确定某个 $x_i$ 所在cache行的相对位置

比如：64B的cache行，存储double变量（8B），可以存储8个，当我们需要确定 $x_i$ 所在cache行的相对位置时，就可以将虚拟地址mod8来进行计算，从而得出相对位置。而当cache行的大小变大时，我们需要动态调整这个mod的数值，比如，当cache行大小变为256B时，存储double变量，可以存储64个，故此时mod的大小就应该为：64。

Our approach is **architecture independent** 架构无关性 因为：

- i) it can be adapted to any cache line size by simply **adjusting this value** before applying the cache-friendly fill-in procedure;
- ii) it can be adapted to any cache indexing mechanism **using virtual addresses**; and, finally,
- iii) it can be applied to **any Instruction Set Architecture (ISA)**.

## Algorithm for Cache-Friendly Fill-In

我们提出了一种算法，用于生成一个通用FSAI稀疏模式的缓存友好的填充。

我们算法的输入是初始稀疏模式 $S$ 和将在SpVM操作中使用的数组 $x$ 。

---

**Algorithm 3** Cache-Friendly Fill-In

---

```
1:  $\mathcal{S} \rightarrow$  Sparse pattern to extend
2:  $x \rightarrow$  Multiplying vector in the SpMV
3: procedure EXTENDPATTERN
4:   Loop over every row  $i$  of pattern  $\mathcal{S}$ 
5:     Loop over every entry  $j$  in row  $i$ 
6:       If  $j$  is in an already considered column block
7:         Move to next  $j$ 
8:       Endif
9:       Identify the cache line of its corresponding  $x_j$  coefficient
10:      Compute initial and final columns matching the cache line of  $x_j$ 
11:      Add to the pattern the columns of the block that are not already present
12:    Endloop
13:  Endloop
```

---

算法3显示了我们的缓存友好填充算法的伪代码。

主循环迭代初始模式行，并且对于每一行，遍历其所有非零条目（行4-13）。

对于每个条目 $j$ ，它通过使用第4.1节中描述的过程来标识相应的 $x_j$ 组件的缓存行（行9），然后通过插入以前不存在的条目来扩展稀疏模式，这些条目对应于存储在同一缓存行中的向量 $x$ 的部分（行10和11）。

该算法可以使用基于线程的方法（如OpenMP或Posix线程）轻松并行化。

## Applying the Cache-Friendly Fill-In to FSAI

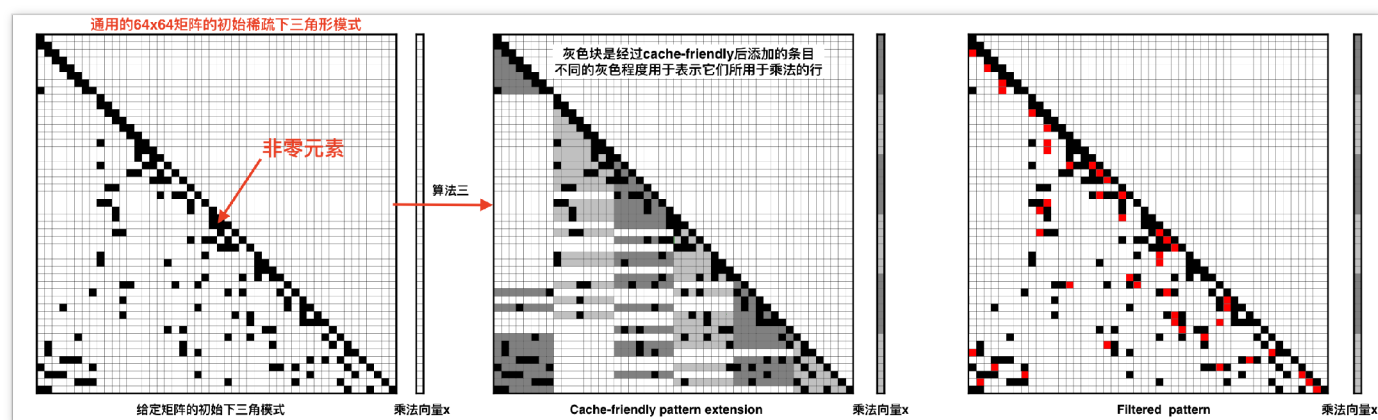
the FSAI pre-conditioner approximates  $A^{-1}$  by the factorization  $G^T G$

The sparse pattern of the original  $G$  matrix is extended using the *algorithm3* to obtain an extended  $G_{ext}$  matrix.

“Therefore, when multiplying by  $G^T ext$ , the entries generated by the extension are accessed in consecutive rows. In conclusion, the spatial locality optimization for the  $G_{ext}$  product results in a temporal locality optimization of the  $G^T ext$  product.” ([Laut 等, 2021, p. 4](#)) ([pdf](#))

☆☆因此，当通过 $G_{ext}^T$ 进行乘法运算时，扩展产生的条目是按连续行访问的。总之，对于 $G_{ext}$ 乘积的空间局部性优化导致了 $G_{Ext}^T$ 乘积的时间局部性优化。

## Cache-Friendly Fill-In Process Example



## Filtering Out Small G Entries

为了使得得到的近似结果更加高效，一种比较常见的方法是通过滤除一些在G中绝对值较小的条目。

但这个方法虽然可以使得近似结果计算更加高效，但是它同样也降低了该前处理器的数字质量。

而且，通过过滤掉小的条目，该稀疏矩阵状态即被修改，所以得到的式子并不一定可以最小化式子 $\min_{G \in S} \|I - GL\|_F^2$

本篇文章提出了一个过滤的新的方法，新方法基于对G的近似预先计算。To generate this low-cost approximation, we solve Eq. 3（上面那个最小化的式子）via several iterations of the CG method with a relatively high tolerance to obtain an approximate solution.

这个cache-friendly extension和 the new filtration 方法 可以用于任何一个给定的稀疏状态

在任何情况下，在利用本文提到的优化方法对稀疏矩阵进行优化时，基本的优化过程如下：

1. 利用cache-friendly 方法来对给定的稀疏模式进行扩展。此过程对应于算法2的第三行。

**Algorithm 2** FSAI,  $G^T G \approx A^{-1}$  with pattern extension and precalculation

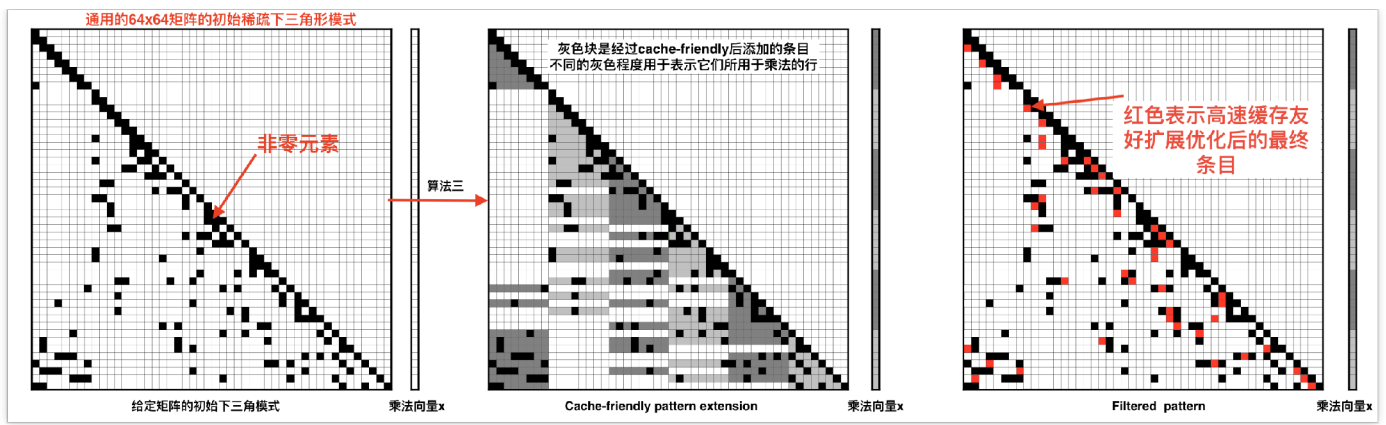
- 1: Threshold  $A$  to produce  $\tilde{A}$ .
- 2: Compute the pattern  $\tilde{A}^N$ , and let the pattern,  $S$ , of  $G$  be the lower triangular part of the pattern of  $\tilde{A}^N$ .
- 3: **Compute cache-friendly extension of the pattern of  $G$ ,  $S_{ext}$ .**
- 4: **Precalculate an approximation  $\tilde{G}$  of the preconditioner, and filter out entries of  $S_{ext}$  according to its values.**
- 5: Calculate  $G$  on the sparse pattern obtained from the previous step.

2. 对扩展模式上的G进行预计算
3. 过滤掉具有较低绝对值特征的条目
4. 使用  $\min_{G \in S} \|I - GL\|_F^2$  方程来计算上面步骤生成的稀疏模式G的系数。该步骤对应于算法2的第五行。

**Algorithm 2** FSAI,  $G^T G \approx A^{-1}$  with pattern extension and precalculation

- 1: Threshold  $A$  to produce  $\tilde{A}$ .
- 2: Compute the pattern  $\tilde{A}^N$ , and let the pattern,  $S$ , of  $G$  be the lower triangular part of the pattern of  $\tilde{A}^N$ .
- 3: **Compute cache-friendly extension of the pattern of  $G$ ,  $S_{ext}$ .**
- 4: **Precalculate an approximation  $\tilde{G}$  of the preconditioner, and filter out entries of  $S_{ext}$  according to its values.**
- 5: Calculate  $G$  on the sparse pattern obtained from the previous step.

这些步骤应用于一个64x64的稀疏矩阵。过程如下图所示。



左侧图像显示了初始矩阵的下三角部分。

在中间的图中，我们展示了模式如何在假设缓存大小为64B和以双精度存储的条目下成为缓存友好的扩展。

右侧图片中的红色部分显示了高速缓存友好扩展的最终条目，而初始条目以黑色表示。

请注意，不同的缓存大小会导致不同的扩展模式。在这个扩展模式上预先计算出了一个逆近似值。在这个扩展模式上预先计算了反向逼近。

## Exploiting Spatial and Temporal Locality in the $G_P$ and $G_P^T$ Products

本节主要介绍利用 $G_P$ 和 $G_P^T$ 提高空间局部性和时间局部性的方法。

前面的部分主要介绍的是如何扩展稀疏模式矩阵 $G$ 从而得到一个扩展的稀疏模式矩阵 $G_{ext}$ ，这个扩展的稀疏模式矩阵，在与向量 $p$ 进行向量乘法时候，导致缓存未命中的次数不会增加。这个扩展基于 $G$ 的空间局部性进行优化，并产生 $G^T$ 的时间局部性优化。

提高FSAI的时间局部性和空间局部性方法如下：

1. 将初始的下三角状态下的矩阵 $S$ 进行cache-friendly扩展，扩展到缓存友好的条目，从而优化 $G$ 乘积中对 $x$ 的访问。



2. 提前计算扩展模式上的近似值
3. 过滤掉扩展位置的条目，从而得到 $S_{ext}$
4. 将 $(S_{ext})^T$  利用cache-friendly进行扩展，得到扩展条目，从而优化 $G^T$ 乘积中x的访问。
5. 提前计算扩展后的近似值
6. 过滤掉扩展位置的条目，从而得到 $((S_{ext})^T)_{ext}$
7. 使用转置的稀疏状态（模式）矩阵 $((S_{ext})^T)_{ext}$  来计算最终的 $G$

具体的算法步骤如下所示：

---

**Algorithm 4** FSAI,  $G^T G \approx A^{-1}$  with pattern extension and precalculation optimizing spatial and temporal locality for both products

---

- 1: Threshold  $A$  to produce  $\tilde{A}$ .
  - 2: Compute pattern  $\tilde{A}^N$ , and define the initial pattern  $\mathcal{S}$  of  $G$  as the lower triangular part of the pattern of  $\tilde{A}^N$ .
  - 3: Compute cache-friendly extension of pattern  $\mathcal{S}$  optimizing  $Gp$  product.
  - 4: Precalculate the nonzero entries in the obtained cache-friendly pattern and keep entries larger than parameter *filter*. Define  $S_{ext}$  as the sparse pattern of the extended matrix  $G_{ext}$ .
  - 5: Compute cache-friendly extension of  $S_{ext}^T$  optimizing  $G_{ext}^T p$  product.
  - 6: Precalculate the nonzero entries in the obtained cache-friendly pattern and keep entries larger than parameter *filter*. Define  $(S_{ext})_{ext}^T$  as the sparse pattern of matrix  $G^T$ .
  - 7: Compute  $G$  coefficients on the transposed pattern  $(S_{ext})_{ext}^T$ .
- 

## Evaluation

### 实验设置

评估考虑到了最先进的FSAI方法，以及使用缓存友好方法扩展稀疏模式的另外两个预处理器。

1. FSAI - Factorized Sparse Approximate Inverse pre-conditioner.

•

---

**Algorithm 1** FSAI,  $G^T G \approx A^{-1}$ 

---

- 1: Threshold  $A$  to produce  $\tilde{A}$ .
  - 2: Compute the pattern  $\tilde{A}^N$ , and let the pattern of  $G$  be the lower triangular part of the pattern of  $\tilde{A}^N$ .
  - 3: Compute the nonzero entries in  $G$  by solving the Frobenius minimization problem.
  - 4: Drop small entries in new  $G$  and rescale.
- 

• 不进行阈值处理并且仅仅对空白目录进行过滤

2. FSAIE(sp) - Factorized Sparse Approximate Inverse preconditioner with a sparse pattern extension exploiting spatial locality.

•

---

**Algorithm 4** FSAI,  $G^T G \approx A^{-1}$  with pattern extension and precalculation optimizing spatial and temporal locality for both products

---

- 1: Threshold  $A$  to produce  $\tilde{A}$ .
  - 2: Compute pattern  $\tilde{A}^N$ , and define the initial pattern  $\mathcal{S}$  of  $G$  as the lower triangular part of the pattern of  $\tilde{A}^N$ .
  - 3: Compute cache-friendly extension of pattern  $\mathcal{S}$  optimizing  $Gp$  product.
  - 4: Precalculate the nonzero entries in the obtained cache-friendly pattern and keep entries larger than parameter *filter*. Define  $\mathcal{S}_{ext}$  as the sparse pattern of the extended matrix  $G_{ext}$ .
  - 5: Compute cache-friendly extension of  $\mathcal{S}_{ext}^T$  optimizing  $G_{ext}^T p$  product.
  - 6: Precalculate the nonzero entries in the obtained cache-friendly pattern and keep entries larger than parameter *filter*. Define  $(\mathcal{S}_{ext})_{ext}^T$  as the sparse pattern of matrix  $G^T$ .
  - 7: Compute  $G$  coefficients on the transposed pattern  $(\mathcal{S}_{ext})_{ext}^T$ .
- 

• 在算法4中不包含第5和第6步骤

- 在第一次乘法时只用到空间局部性，在第二次乘法时，用到时间局部性

### 3. FSAIE(full) -Factorized Sparse Approximate Inverse preconditioner with pattern extension exploiting spatial and temporal locality.

- 算法4
- 在所有的乘法中，都要用到空间局部性和时间局部性。

## Performance Improvement on Skylake

这部分介绍了FSAIE（sp）和FSAIE（full）相对于FSAI的性能。

ID	Matrix	#rows	NNZ	Type	FSAI			FSAIE(sp)				FSAIE(full)			
					Setup	Solve	Iter	Setup	Solve	Iter	% NNZ	Setup	Solve	Iter	% NNZ
1	shipsec5	179860	4598604	Structural	9.58E-02	1.08E+00	1615	3.01E-01	1.01E+00	1465	14.95	5.39E-01	1.04E+00	1437	20.82
2	offshore	259789	4242673	Electromagnetics	8.91E-02	8.97E-01	782	2.82E-01	8.25E-01	771	16.39	6.82E-01	9.07E-01	751	30.86
3	smt	25710	3749582	Structural	3.14E-01	4.32E-01	884	9.78E-01	3.06E-01	574	20.49	1.78E+00	2.95E-01	515	33.19
4	parabolic_fem	525825	3674625	CFD	6.45E-02	2.26E+00	1460	2.16E-01	1.87E+00	1144	65.78	5.31E-01	1.67E+00	1054	119.98
5	Dubcova3	146689	3636643	2D/3D	7.36E-02	1.19E-01	153	2.99E-01	1.46E-01	138	62.55	5.89E-01	1.22E-01	107	110.01
6	shipsec1	140874	3568176	Structural	8.53E-02	1.10E+00	1985	2.26E-01	1.09E+00	1982	14.49	4.39E-01	1.12E+00	1945	19.49
7	nd3k	9000	3279690	2D/3D	1.75E+00	1.97E-01	406	4.19E+00	1.76E-01	357	2.07	6.05E+00	1.65E-01	336	3.03
8	cfid2	123440	3085406	CFD	5.81E-02	1.21E+00	2600	2.01E-01	1.14E+00	1969	74.60	4.17E-01	1.21E+00	1862	120.11
9	nasasrb	54870	2677324	Structural	8.90E-02	1.10E+00	2768	1.96E-01	1.11E+00	2761	5.95	2.92E-01	1.10E+00	2739	8.87
10	oilpan	73752	2148558	Structural	5.94E-02	5.85E-01	1620	1.24E-01	5.53E-01	1452	24.81	2.22E-01	5.36E-01	1326	47.70
11	cfid1	70656	1825580	CFD	4.74E-02	3.56E-01	932	1.49E-01	3.33E-01	801	62.48	3.15E-01	3.41E-01	739	113.35
12	qa8fm	66127	1660579	Acoustics	4.12E-02	4.14E-03	13	9.71E-02	3.58E-03	11	22.44	1.68E-01	3.57E-03	11	28.70
13	2cubes_sphere	101492	1647264	Electromagnetics	4.74E-02	5.60E-03	12	1.24E-01	5.74E-03	12	10.20	2.50E-01	5.34E-03	11	17.30
14	thermomech_dM	204316	1423116	Thermal	6.11E-02	5.80E-03	9	9.11E-02	5.85E-03	9	2.08	1.97E-01	5.79E-03	9	2.42
15	msc10848	10848	1229776	Structural	1.85E-01	2.18E-01	712	3.50E-01	2.04E-01	651	8.36	6.03E-01	1.64E-01	528	21.51
16	Dubcova2	65025	1030225	2D/3D	4.31E-02	6.04E-02	158	1.08E-01	5.60E-02	131	88.81	2.15E-01	4.90E-02	106	162.91
17	gyro	17361	1021159	Model Reduction	7.22E-02	1.72E+00	4457	1.55E-01	1.11E+00	3576	25.93	2.67E-01	1.38E+00	3400	35.16
18	gyro_k	17361	1021159	DMR	7.30E-02	1.54E+00	4444	1.49E-01	1.24E+00	3599	25.93	2.60E-01	1.02E+00	3450	35.16
19	olafu	16146	1015156	Structural	5.42E-02	4.17E-01	1782	1.10E-01	3.80E-01	1524	14.00	1.71E-01	3.15E-01	1336	22.64
20	bundle1	10581	770811	CG/V	1.08E-01	6.82E-03	22	2.11E-01	5.83E-03	20	0.01	3.00E-01	6.04E-03	20	0.01
21	G2_circuit	150102	726674	Circuit Simulation	3.25E-02	3.84E-01	1026	6.69E-02	3.23E-01	808	146.97	1.11E-01	3.18E-01	772	215.71
22	Pres_Poisson	14822	715804	CFD	5.85E-02	6.53E-02	285	9.99E-02	3.89E-02	160	35.75	1.62E-01	3.23E-02	130	61.49
23	thermomech_TC	102158	711558	Thermal	3.50E-02	3.94E-03	9	7.83E-02	3.90E-03	9	3.13	1.14E-01	3.96E-03	9	3.65
24	cbuckle	13681	676515	Structural	5.07E-02	2.48E-02	114	8.83E-02	2.55E-02	108	12.55	1.38E-01	2.30E-02	101	24.08
25	finan512	74752	596992	Economic	3.12E-02	2.88E-03	10	5.95E-02	2.72E-03	9	34.16	8.94E-02	2.79E-03	9	42.53
26	crystm03	24696	583770	Materials	2.91E-02	3.45E-03	13	6.04E-02	3.51E-03	12	14.04	9.75E-02	3.26E-03	11	26.34
27	thermal1	82654	574458	Thermal	3.19E-02	2.80E-01	735	5.85E-02	2.24E-01	603	95.58	1.34E-01	2.26E-01	532	189.89
28	wathen120	36441	565761	Random 2D/3D	2.97E-02	6.10E-03	25	5.20E-02	4.88E-03	19	76.92	8.82E-02	4.85E-03	19	98.41
29	apache1	80800	542184	Structural	2.89E-02	4.43E-01	1663	4.46E-02	4.17E-01	1582	66.59	6.92E-02	4.32E-01	1574	73.41
30	gridgena	48962	512084	Optimization	3.06E-02	4.32E-01	1729	4.39E-02	3.49E-01	1350	80.29	8.16E-02	3.23E-01	1205	141.49

表1显示了Skylake系统中使用三种技术和过滤器= 0.01的结果。

第6至8列报告了FSAI的设置时间、求解时间和收敛所需的迭代次数。

这些是我们用来比较我们的模式扩展方法的Skylake基准结果。

第9至12列分别报告了设置时间、求解时间、收敛迭代次数以及FSAIE（sp）添加到A的下三角形模式中的条目百分比。第13至16列报告了关于FSAIE（full）的这些相同度量指标。

“many cases we observe both FSAIE(sp) and FSAIE(full) to successfully **decrease iteration count and solve time**. FSAIE(full) obtains larger pattern extensions than FSAIE(sp), which **produces larger iteration count and solution time decreases**.”([pdf](#))

“In all cases, our new filtration strategy **avoids convergence degradation** while providing a higher degree of robustness to the method” ([pdf](#))

## Effects on Data Cache Misses and FLOPS

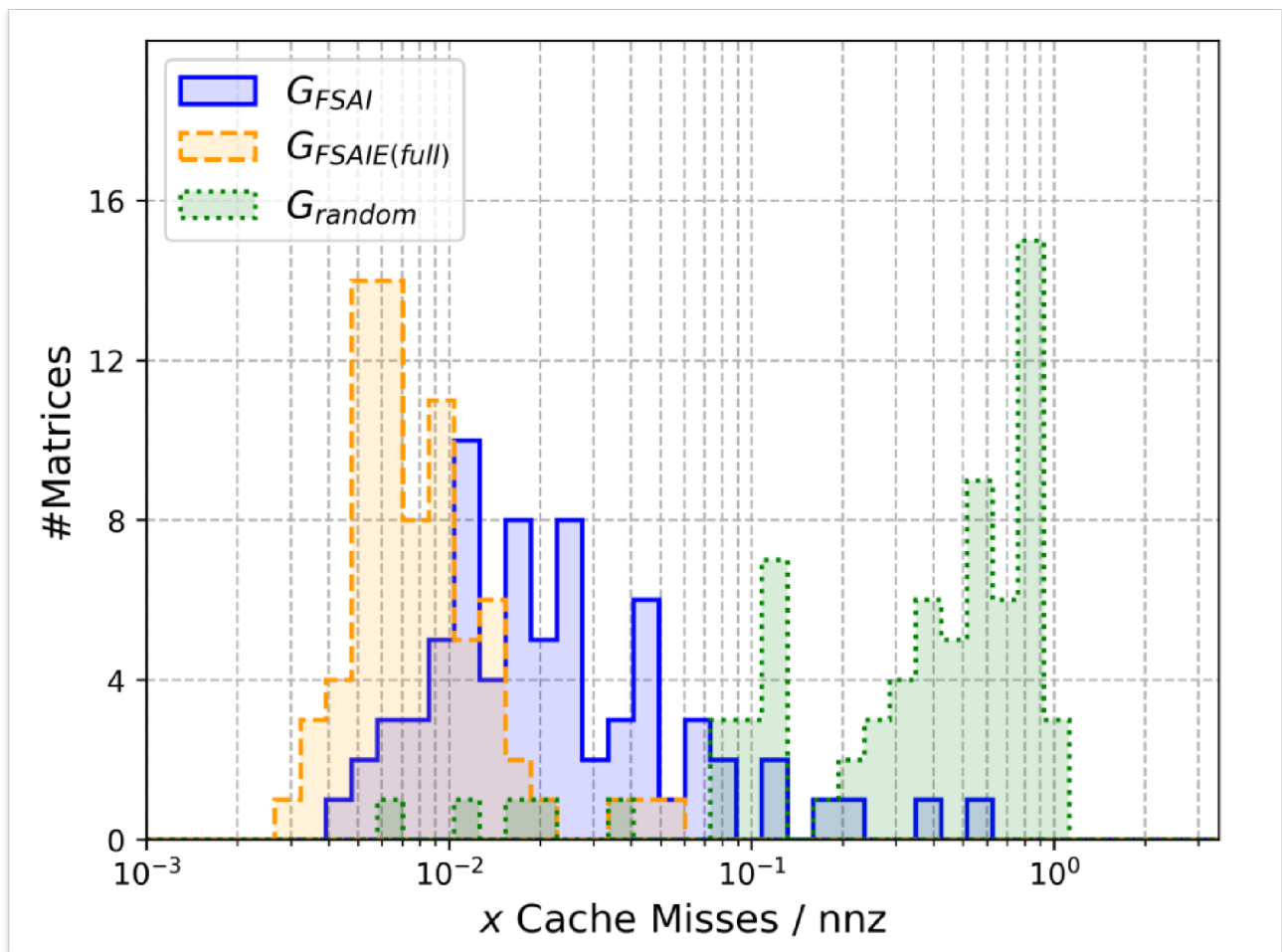
该部分描述了在Skylake上使用其扩展的稀疏模式时，FSAIE（full）方法在数据缓存未命中和每秒浮点操作（flop/s）方面的优势。

FSAIE（FULL）在两个方面上提高了效率：

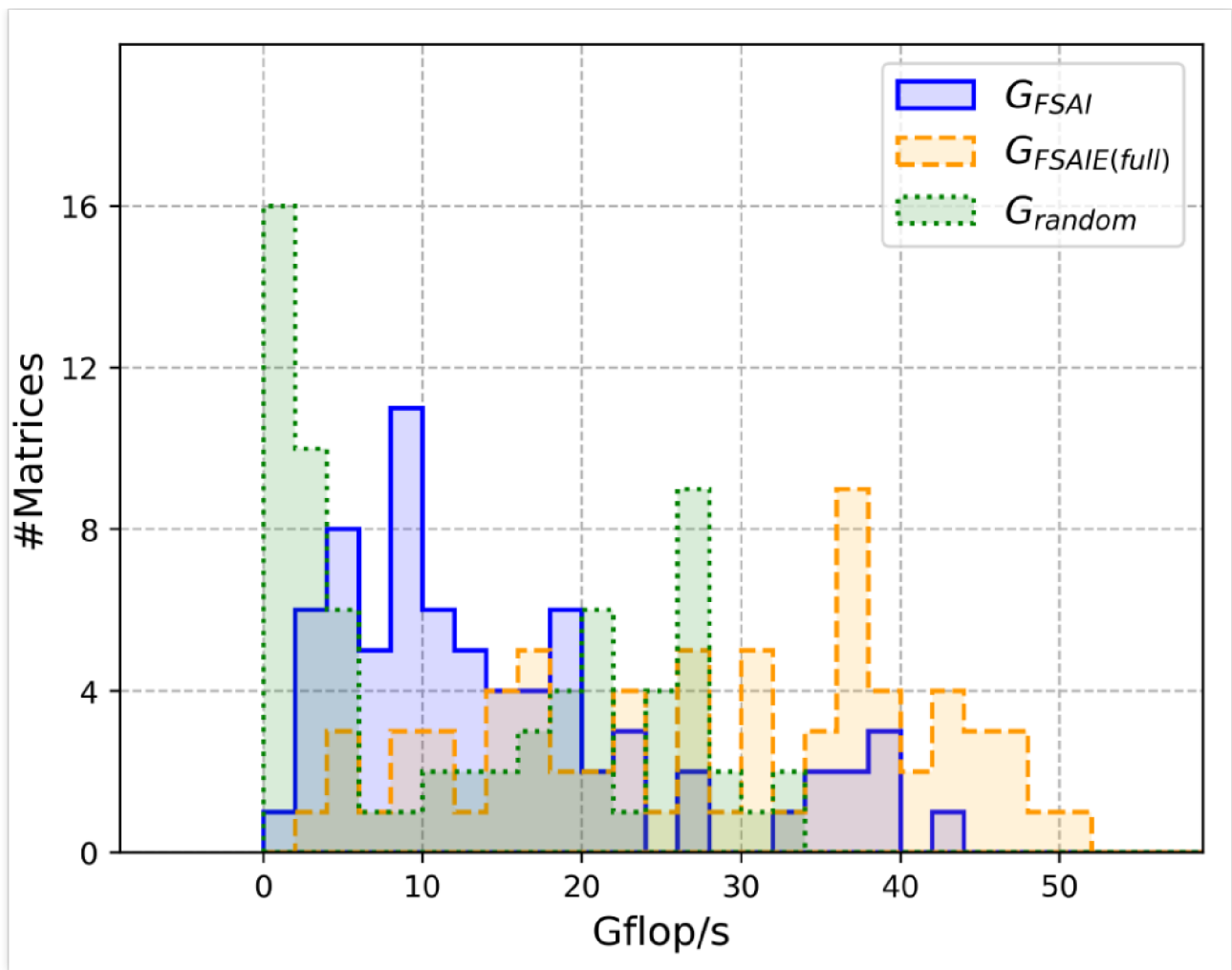
- 首先，通过扩展预处理稀疏模式来实现迭代次数的减少；
- 其次，保持使用了模式扩展之后带来的额外迭代成本仍然处于较低水平。

7.2节展示了相对于Skylake系统中的FSAI而言，FSAIE(full)在解决时间和迭代次数方面的优势。

这一节展示了FSAIE(full)在第二个方面的效果。



“These results clearly indicate how our cache-friendly sparse pattern extensions successfully **minimize the average data cache misses per  $G$  non-zero entry.**”



“Figure 4 clearly shows how the cache-aware extensions of the FSAIE(full) method significantly **improve the floating-point throughput achieved** by the sparse patterns of baseline FSAI.”

## Setup Phase Overhead

尽管该文章提出的方法显著加速了共轭梯度法的求解阶段，但相对于FSAI，在稀疏模式的扩展阶段会带来一些**额外开销**，这主要是由于**计算G矩阵条目的成本更高**。

**Such overhead becomes negligible in a practical numerical simulation context**

- since the **setup phase** is performed **only once** while the **solve phase is repeated** several times for the same matrix.
- Furthermore, even when the setup is to be repeated on each time step, some of its parts, such as the definition of the final pattern, do not need to

be repeated on each time step.

## Evaluation on POWER9

FSAIE(full)				
Filter value	Avg. iterations	Avg. time	Highest imp.	Highest deg.
0.0	18.55	-14.24	52.26	-208.93
0.001	17.96	2.49	54.49	-58.08
0.01	16.90	10.25	56.72	-18.90
0.1	8.99	8.56	42.75	-12.35
Best filter	15.15	12.94	56.72	-12.35

“For most of matrices  $f_{ilter} = 0.01$  is the best option.” ([Laut 等, 2021, p. 10](#)) ([pdf](#))

## Evaluation on A64FX

FSAIE(full)				
Filter value	Avg. iterations	Avg. time	Highest imp.	Highest deg.
0.0	27.81	-17.52	76.99	-575.19
0.001	26.47	14.93	63.47	-54.79
0.01	23.98	20.08	61.38	-6.58
0.1	13.36	13.76	48.03	-3.80
Best filter	24.91	22.85	76.99	-0.96

这些大幅度的迭代次数减少带来了明显的性能改善。

FSAIE（完全）方法在 $filter = 0.01$ 时带来了平均性能提升20.08%，使用最佳 $filter$  per matrix时为22.85%。

不过滤任何条目，即 $filter = 0.0$ ，性能会下降，因为迭代成本增加超过了迭代次数的减少。

We show the results of the best performing  $f\ filter$  value (blue columns) for each matrix and the results for the best general  $f\ filter$  value (orange columns), **which is 0.01**. Many matrices display much larger performance improvements on A64FX than POWER9 and Skylake.

## Comparing Results on Different Architectures

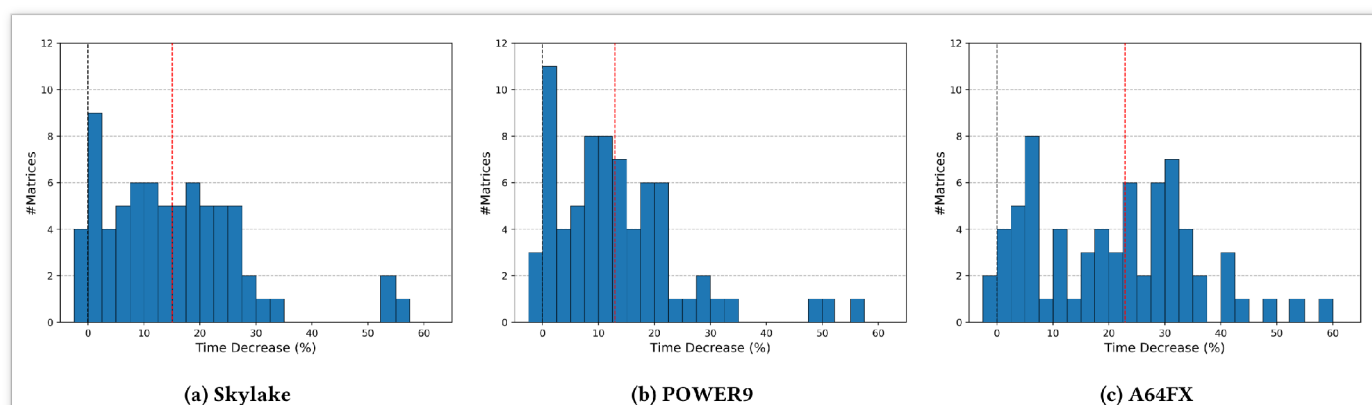


Figure 7: Histograms classifying the 72 matrices of our experimental set in terms of time improvement achieved by FSAIER(full) using the best  $f\ filter$  value with respect to FSAI.

虽然 Skylake 和 POWER9 的结果显示出类似的趋势，因为它们具有基本相同的模式扩展，但当 FSAIE (full) 应用于 A64FX 时，它获得了更丰富的稀疏模式，这显著提高了实验集中所有矩阵的平均改善。

这是因为 A64FX 的缓存行大小比 Skylake 或 POWER9 大4倍。

## Related work on approximate inverse methods

有几种先前提出的方法来为稀疏近似逆问题生成模式。根据稀疏近似逆的模式如何评估，它们被认为是静态或动态方法。



# Static approach

对于静态方法（本文考虑的方法），模式是事先确定的，并在逆近似计算期间保持不变，无论是M还是G的分解方法。

常见的一种是使用矩阵A的稀疏模式的幂次方，通常是 $A^2$ 或 $A^3$  (10, 16, 20)。

其他技术会重新塑造初始模式(23)。

通过**阈值化**和**后处理过滤**或**自适应入口丢弃策略**(5, 6, 11, 15, 27, 29)可以将生成的模式变为**稀疏模式**。

寻找最佳的阈值化和过滤准则通常是一项具有挑战性的任务。

# Dynamic approximate inverse methods

动态的近似逆方法通过自适应程序从初始猜测（例如对角线模式）开始，并使用某种策略扩大该模式，直到满足特定的标准，从而计算出逆模式。

An example of a dynamic approach, SPAI, was proposed by Grote and Huckle [17]. There is also a factorized formulation, FSPAI [21]. Other dynamic strategies have been developed more recently, such as, its generalization to block form, BSAI [22], and others like PSAI and RSAI [25, 26].

通常，动态近似逆比其静态的方法更为有效。然而，高效地将它们并行化并且它们的预处理阶段通常比静态方法更昂贵并非易事。

# 总述

所有引用的近似逆方法，无论是静态的还是动态的，共同的一个因素是它们都没有考虑到准则来定义稀疏模式。

基于缓存友好模式扩展的概念，我们的方法与提到的任何替代方法互补。最常见的静态方法FSAI在这里被用作参考。然而，无论使用何种其他具有数值评估标准的模式，我们的方法都能带来潜在的显著性能提升。

# Conclusions

This paper demonstrates the benefits of a **FSAI sparse pattern extension** based on two fundamental concepts:

- 一个能够生成缓存感知的稀疏模式扩展的算法，该算法可以显著降低CG迭代次数并且带有较低迭代时间开销。
- 一个强大的过滤策略，最大程度地利用了缓存感知扩展的好处。

尽管最先进的方法**仅基于数值考虑来定义稀疏模式**，但本文首次展示了考虑计算机体系结构概念的好处。

实际上，所提出的缓存感知模式扩展是对用于定义稀疏模式的任何数值策略的补充。