Contents

Project description

To describe the services an operating system provides to users, processes, and other systems
To discuss the various ways of structuring an operating system
To explain how operating systems are installed and customized and how they boot

Introduction

Design

appendix

https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=AVX_512&cats=Eleme

https://www.codeproject.com/Articles/874396/Crunching-Numbers-with-AVX-and-AVX

_mm<bit_width>_<name>_<data_type>

The parts of this format are given as follows:

<bit_width> identifies the size of the vector returned by the function. For 128-bit vect

<name> describes the operation performed by the intrinsic
<data_type> identifies the data type of the function's primary arguments

Data Type	Description
__m128i	128-bit vector containing 8 integers
__m128	128-bit vector containing 4 floats
__m128d	128-bit vector containing 2 doubles
__m256i	256-bit vector containing 16 integers
__m256	256-bit vector containing 8 floats
__m256d	256-bit vector containing 4 doubles
__m512i	512-bit vector containing 32 integers
__m512	512-bit vector containing 16 floats
__m512d	512-bit vector containing 8 doubles

ps - vectors contain floats (ps stands for packed single-precision)
pd - vectors contain doubles (pd stands for packed double-precision)
epi8/epi16/epi32/epi64 - vectors contain 8-bit/16-bit/32-bit/64-bit signed integers
epu8/epu16/epu32/epu64 - vectors contain 8-bit/16-bit/32-bit/64-bit unsigned integers
si128/si256 - unspecified 128-bit vector or 256-bit vector
m128/m128i/m128d/m256/m256i/m256d - identifies input vector types when they're different

notes from lecture with Jakob

Our cache use, calculate it to be = to 256 or maybe a bit less. $kcandmcshouldbelarge.n_cislessimportant.$
kc = 128 MC = 128, $8 * (m_c * k_c + m_c * n_c + N_c + k_c)$
$avx_2 56 registersarecalled ymm 0 to 15$ $avx_5 12 registersarecalled zmm 0 to 31$
Make function for each size of slices.. 128 = 4.4 256 = 8.4 512 = 8.8